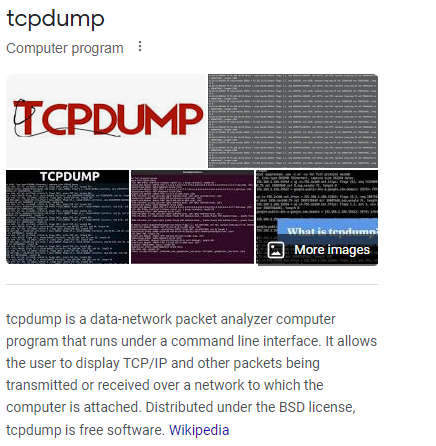
Project Pro 50 - Final Project

# Business Problem:

The raw network packets of the UNSW-NB 15 dataset was created by the IXIA PerfectStorm tool in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) for generating a hybrid of real modern normal activities and synthetic contemporary attack behaviours.



Tcpdump tool is utilised to capture 100 GB of the raw traffic (e.g., Pcap files). This dataset has nine types of attacks, namely, Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms. The Argus, Bro-IDS tools are used and twelve algorithms are developed to generate totally 49 features with the class label.

The Challenge here is there are multiple Algorithms generating lots of data with 49 features and it is a critical application and has to be taken action in Real-Time. But due to the volume of data it is difficult to analyse the data in real-time and take timely action.

So we need to build a robust data pipeline that can read data in real-time and provide a dashboard where we can analyse any alerts and take immediate action.

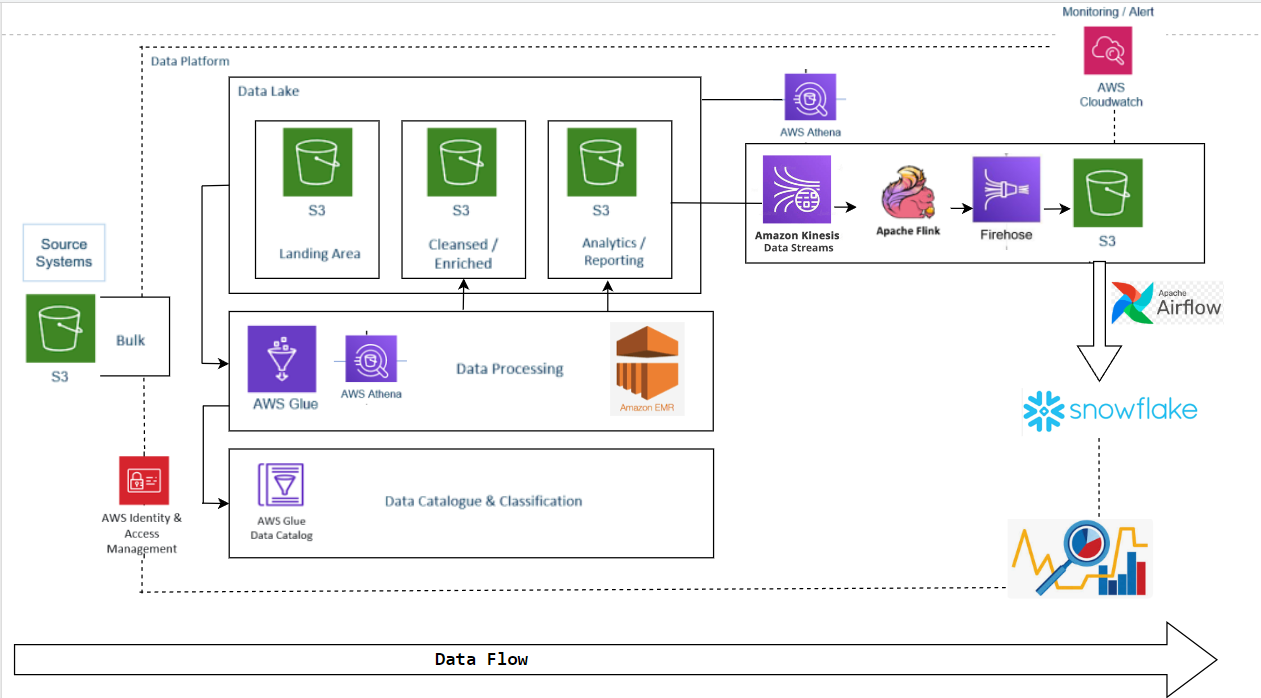
# Data Pipeline Approach:

So we will be receiving the data from the API and receive it in form of updated csv file in AWS S3 bucket. Once the Data lands in S3 we will have to Clean the data by doing some filtrations and we will achieve this using AWS Glue and Aws Lambda and store the transformed data in S3 bucket. Now we will Process the data further in AWS EMR and again stored in a new AWS S3 bucket. The data that is transformed is now Validated through AWS Athena.

Now this is happening in Live so we will have to stream the data to a dashboarding tool and we will achieve this through AWS Kinesis Data Stream from where the data will be read through Kinesis Firehouse that could update Snowflake Tables. The whole pipeline of Streaming is automated using Airflow.

From Snowflake data can be connected to dashboarding tools and visualized in real time. The issues that might arise here is Permissions to handle all the AWS Data Services and concerns with the scripts that we can trouble shoot with AWS cloud watch. The Data Flow pipeline architecture is drafted below.

# Data Pipeline Architecture:



# Steps in the Project:

1. Load the csv data file in the s3 location: s3://<bucket>/unsw-nb15/raw\_data/
2. Create a new Glue catalog database: "unsw\_nb15". All the tables in this project will be created under this database.
3. Create a new Glue catalog table named "raw\_unsw\_nb15" having 49 columns and data as mentioned in features csv file. (You can assume type "nominal" to be string.)
4. Transform the data into a new table named "proto\_filtered" where "proto" column is having the value of either "tcp" or "udp".
5. The content of this new table should be put into another s3 location on s3://<bucket>/unsw\_nb15/proto\_filtered/" in csv format.
6. Now, firstly we will do batch filtering. For this, create a EMR job that reads the proto filtered data file from S3 and gets those records where "label" = 1 using pyspark, and upload this file into a separate location in s3: s3://<bucket>/unsw-nb15/attack\_records/
7. Create a mapping of this table in Glue, and query this data using Athena. Ensure that you get no records where "label" is 0, and "proto" not in ("tcp", "udp")
8. Now, create a python script to stream the data from attack\_records file into a Kinesis stream.
9. Create flink sql that reads the data from this kinesis stream "raw\_attack\_records", and filters out those records where "service" is either "dns" or "-", and pushes it out to another kinesis stream "dns\_or\_unknown\_attack\_records".
10. Create a kinesis firehose to read from the "dns\_or\_unknown\_attack\_records" kinesis stream, and land it in s3 location: s3://<bucket>/unsw-nb15/dns\_or\_unknown\_attack\_records/
11. We will get this latest processed file into Snowflake schema "unsw-nb15-schema", database "unsw-nb15-db" in the table "attack\_records". Orchestrate this using SnowflakeOperator in Airflow. Once the records are copied into Snowflake table, move the file from s3 location s3://<bucket>/unsw-nb15/dns\_or\_unknown\_attack\_records/ to s3://<bucket>/unsw-nb15/attack\_records\_processed/
12. Now, query the snowflake table to figure out how many attack records are present in each of the attack category represented by the column "attack\_cat".

# Services used for Project:

Resources and Naming conventions that we are using for this project:

1. **AWS S3:**

* s3://p50-bigdata-rawdata/unsw-nb15/raw-data/
* s3://p50-bigdata-proto-filtered/unsw-nb15/proto\_filtered/
* s3://p50-bigdata-filtered-attack-records/unsw-nb15/attack\_records/
* s3://p50-bigdata-dns\_or\_unknown\_attack\_records/unsw-nb15/dns\_or\_unknown\_attack\_records/

1. **AWS IAM:**

* p50-servicerole-01
* p50-servicerole-02
* [kinesis-analytics-p50\_flink\_stream\_studio-sql-flink-de-us-east-2](https://us-east-1.console.aws.amazon.com/iamv2/home?region=us-east-2#/roles/details/kinesis-analytics-p50_flink_stream_studio-sql-flink-de-us-east-2)
* [kinesis-analytics-p50\_flink\_stream\_studio-us-east-2](https://us-east-1.console.aws.amazon.com/iamv2/home?region=us-east-2#/roles/details/kinesis-analytics-p50_flink_stream_studio-us-east-2)
* [KinesisFirehoseServiceRole-p50\_firehose\_-us-east-2-1694385850803](https://us-east-1.console.aws.amazon.com/iamv2/home?region=us-east-2#/roles/details/KinesisFirehoseServiceRole-p50_firehose_-us-east-2-1694385850803)
* [P50\_SNOWFLAKE\_ACCESS\_ROLE](https://us-east-1.console.aws.amazon.com/iamv2/home?region=us-east-2#/roles/details/P50_SNOWFLAKE_ACCESS_ROLE)

1. **AWS Glue:**

* Database: **unsw\_nb15**
* Crawlers
* Tables
* Glue Job: **Pyspark**

1. **AWS EMR :** p50\_emr\_job
2. **AWS Kinesis**

**Kinesis Streams:**

* p50\_raw\_attack\_records\_stream
* p50\_dns\_or\_unknown\_attack\_records\_stream

**Kinesis Firehose:**

* P50\_firehose\_stream

**Flink :**

Studio: Zeplin Notebook for **Kinesis**

* p50\_flink\_stream\_studio
* p50\_flink\_stream\_application

1. **Snowflake**

Database: P50\_FINAL\_DB

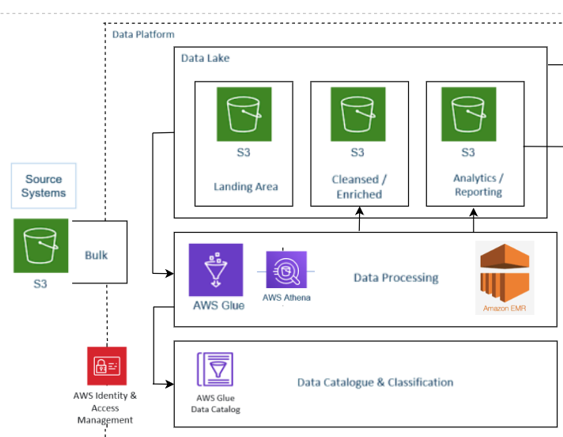
Warehouse: P50\_FINAL\_WAREHOUSE

Schema: P50\_FINAL\_SCHEMA

Table: P50\_FINAL\_TABLE

# Project: Steps and Explanation (3 Phases)

## Phase: 1



### Creating the Buckets as per planned Architecture

Firstly we are creating the necessary S3 buckets for our project according to how we have planned in the architecture and then we will create the necessary sub folders that is needed to cater the Data flow.

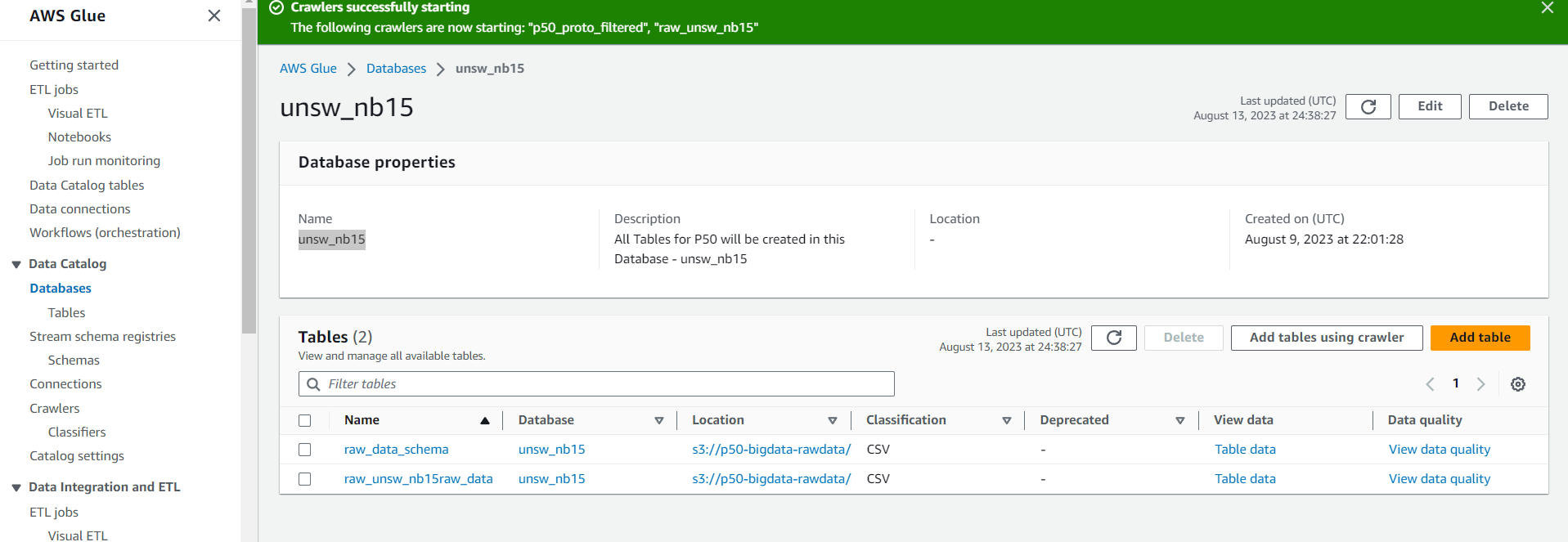
### Loading the Data to AWS S3

Followed by creating the S3 buckets we will load the Dataset (we have sampled the dataset here instead of taking the whole to save compute and time)

S3 Bucket location : s3://p50-bigdata-rawdata/unsw-nb15/raw\_data/UNSW-NB15\_1\_Sample.csv

### (1.3) Creating Database in AWS Glue

Now according to the problem statement all the necessary tables in the project will be created under one database and the name of the database here is : **unsw\_nb15**

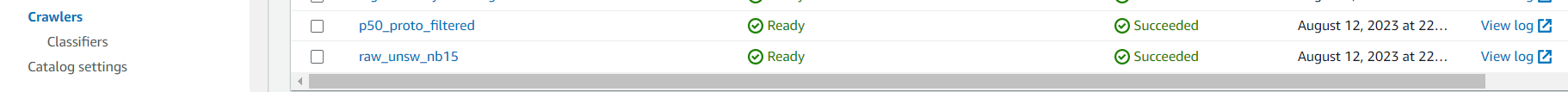
****

### (1.4) Adding a Crawler to parse the Data source and run it

Now we have created a vanilla database and now we will need to create tables into this database and to achieve this we will use the **AWS Glue Crawler** which will parse through the data source identify the schema and load the data into Tables under the respective database.

For this project to read through the Dataset and Data Schema I have configured 2 crawlers:

* [p50\_proto\_filtered](https://us-east-2.console.aws.amazon.com/glue/home?region=us-east-2#/v2/data-catalog/crawlers/view/p50_proto_filtered)
* [raw\_unsw\_nb15](https://us-east-2.console.aws.amazon.com/glue/home?region=us-east-2#/v2/data-catalog/crawlers/view/raw_unsw_nb15)

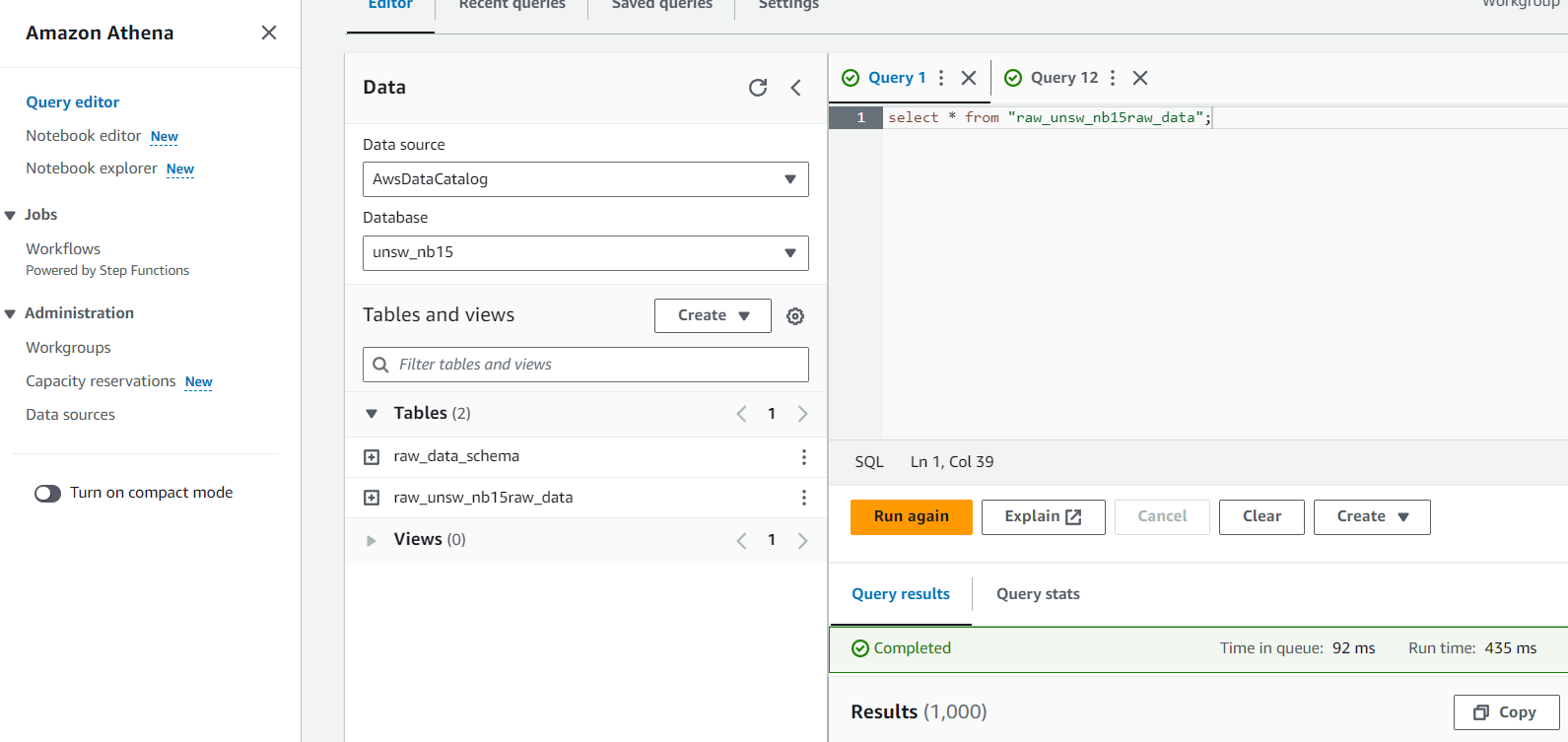


Once the Crawler is setup we should run the crawler and verify if it is functioning as expected. As we see below the crawler should run successfully and generate tables.



### (1.5) Verify Table load through AWS Athena

So we can open the view data under the tables and run Select queries in Athena to confirm the setup. We need to confirm on whether table having 49 columns and data as mentioned in features csv file



### (1.6) Transformation-1 (‘Proto’ filter)

We will Achieve this through Spark. So we will create a Pyspark code that will read the data from the Data Catalog Table and filter out the ‘proto’ column through spark sql and the transformed data will then be staged to a new updated Glue table and S3 bucket.

Steps:

* Create a new table with the schema of the raw data
* Create a pyspark code for transformation in AWS glue ETL jobs
* Run the ETL job and verify if the data is Staged in S3 and New Table
* Create a Crawler to create a new table from the S3 stage
* Verify the results through Athena

import sys

from awsglue.transforms import \*

from awsglue.utils import \*

from pyspark.context import SparkContext

from awsglue.context import GlueContext

from awsglue.dynamicframe import DynamicFrame

from awsglue.job import Job

# Initialize SparkContext

sc = SparkContext()

# Initialize GlueContext

glueContext = GlueContext(sc)

# Initialize SparkSession

spark = glueContext.spark\_session

# Read source table

source\_frame = glueContext.create\_dynamic\_frame.from\_catalog(database = "unsw\_nb15", table\_name = "unsw\_nb15")

# Apply SQL query

query\_results\_frame = source\_frame.toDF().createOrReplaceTempView("temp\_table")

query\_results = spark.sql("SELECT \* FROM temp\_table WHERE proto IN ('tcp', 'udp')")

# Create a new DynamicFrame

proto\_filtered\_data = DynamicFrame.fromDF(query\_results, glueContext, "proto\_filtered\_data")

glueContext.write\_dynamic\_frame.from\_catalog(frame = proto\_filtered\_data, database = "unsw\_nb15", table\_name = new\_table\_name)

# Repartition the data to a single output file (optional)

proto\_filtered\_data = proto\_filtered\_data.repartition(1)

# Write the filtered data to an S3 bucket in CSV format

output\_s3\_path = "s3://p50-bigdata-proto-filtered/unsw-nb15/proto\_filtered/"

glueContext.write\_dynamic\_frame.from\_options(

    frame=proto\_filtered\_data,

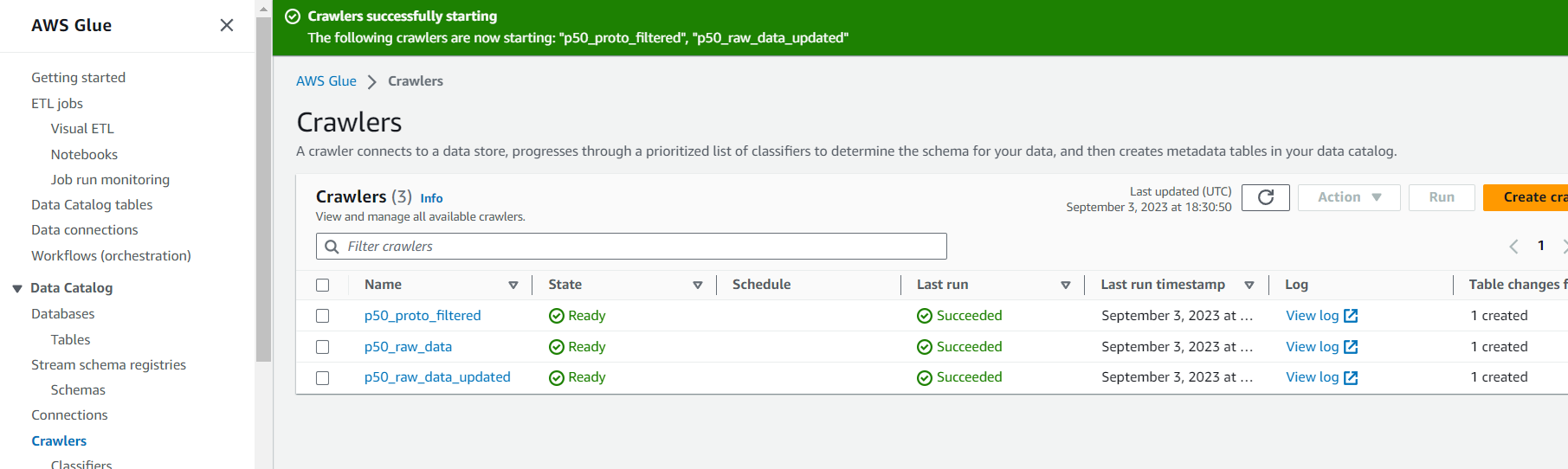
    connection\_type="s3",

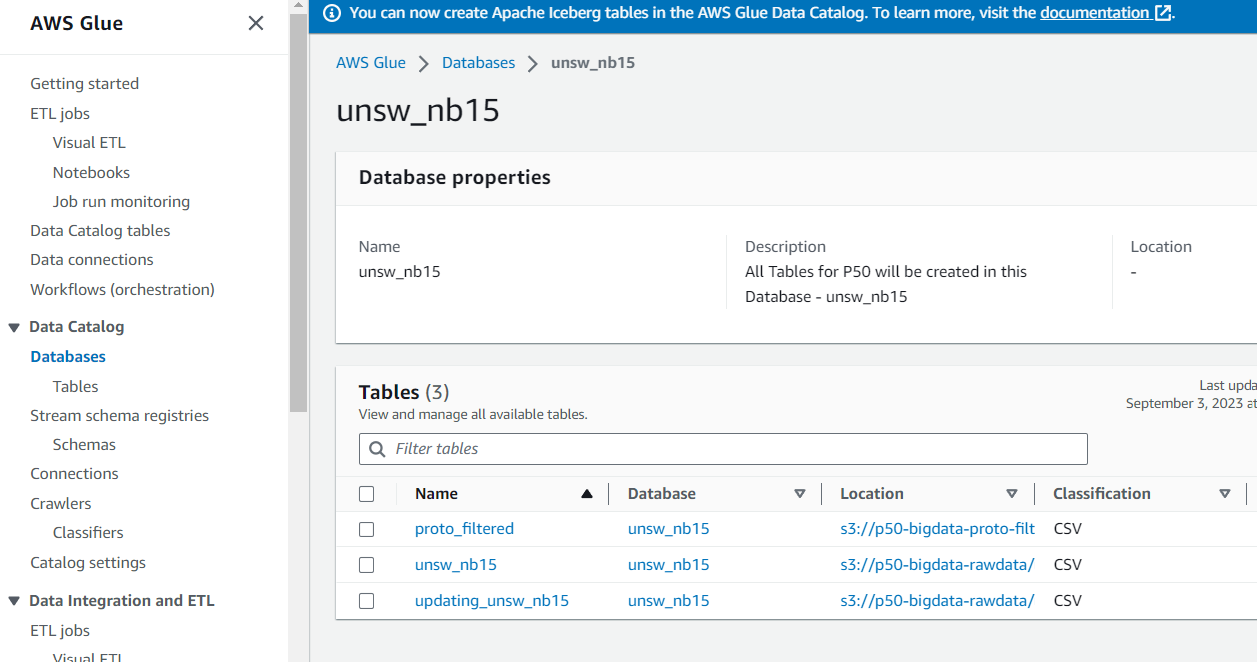
    connection\_options={"path": output\_s3\_path},

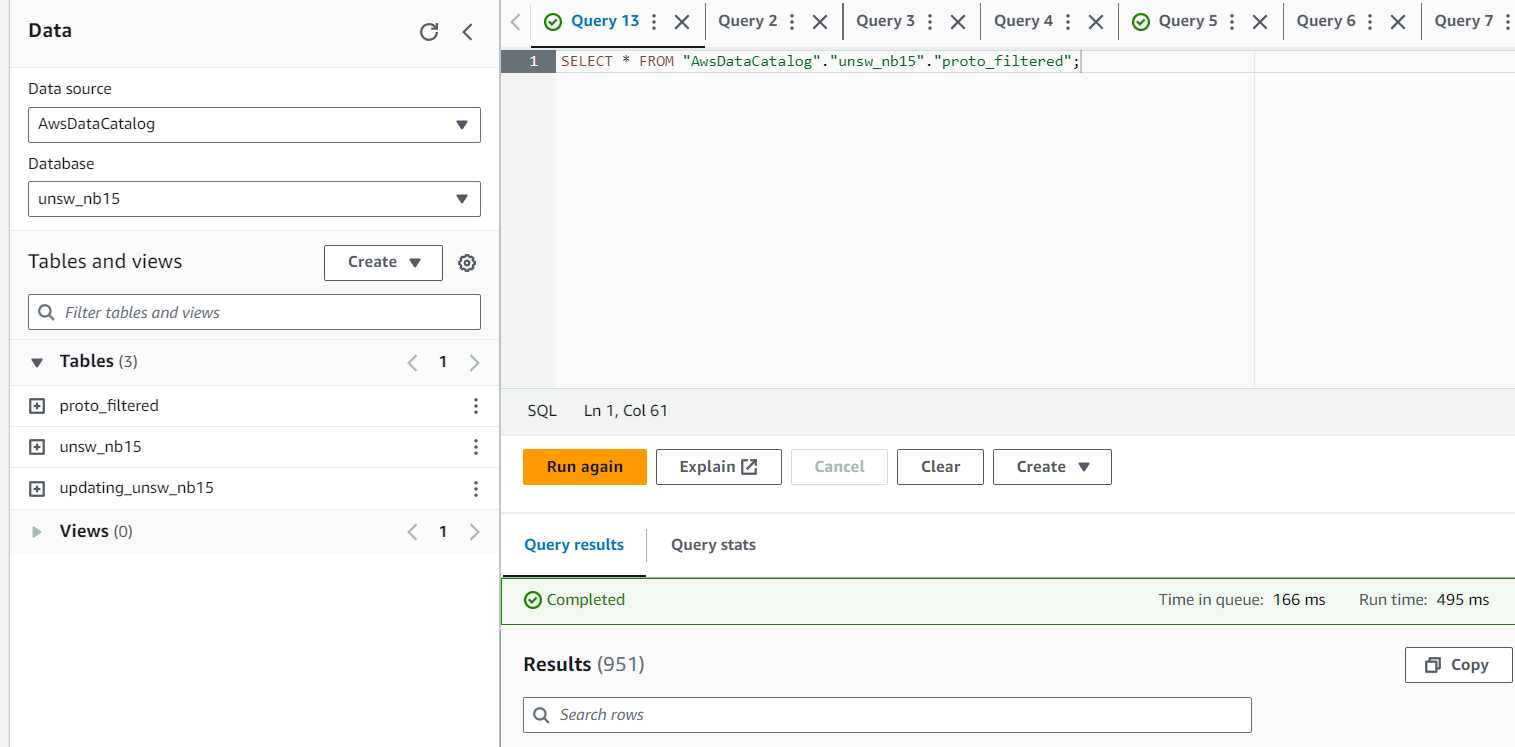
    format="csv",

    format\_options={"writeHeader": True}  # Optional, write column headers in CSV

)



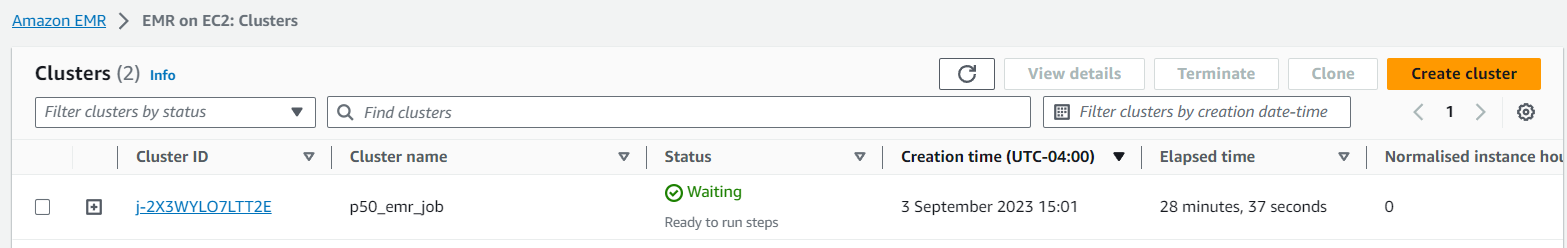




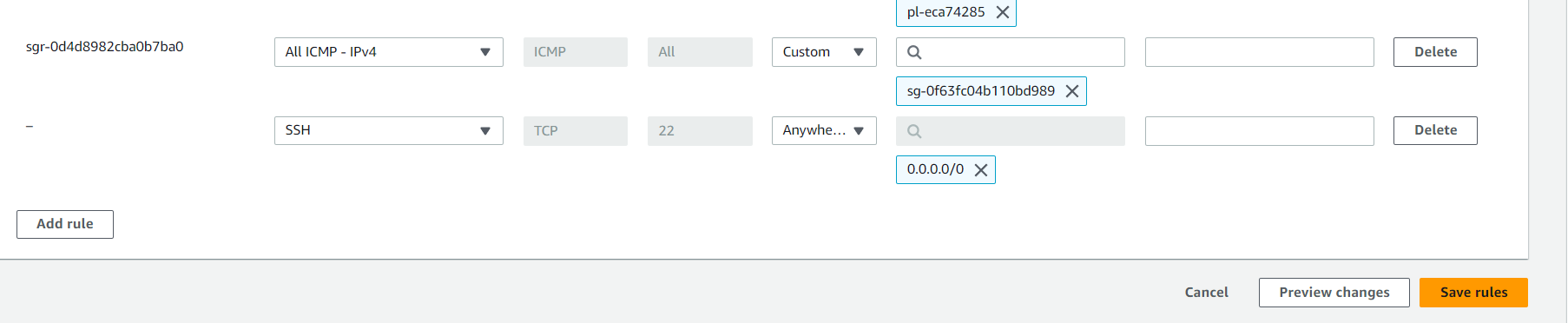
### (1.7) Transformation-2 (EMR job for Attack Records)

We will now create a EMR cluster and then create a Pyspark job to do Data transformation and store the transformed data in S3.

**EMR Cluster Created**

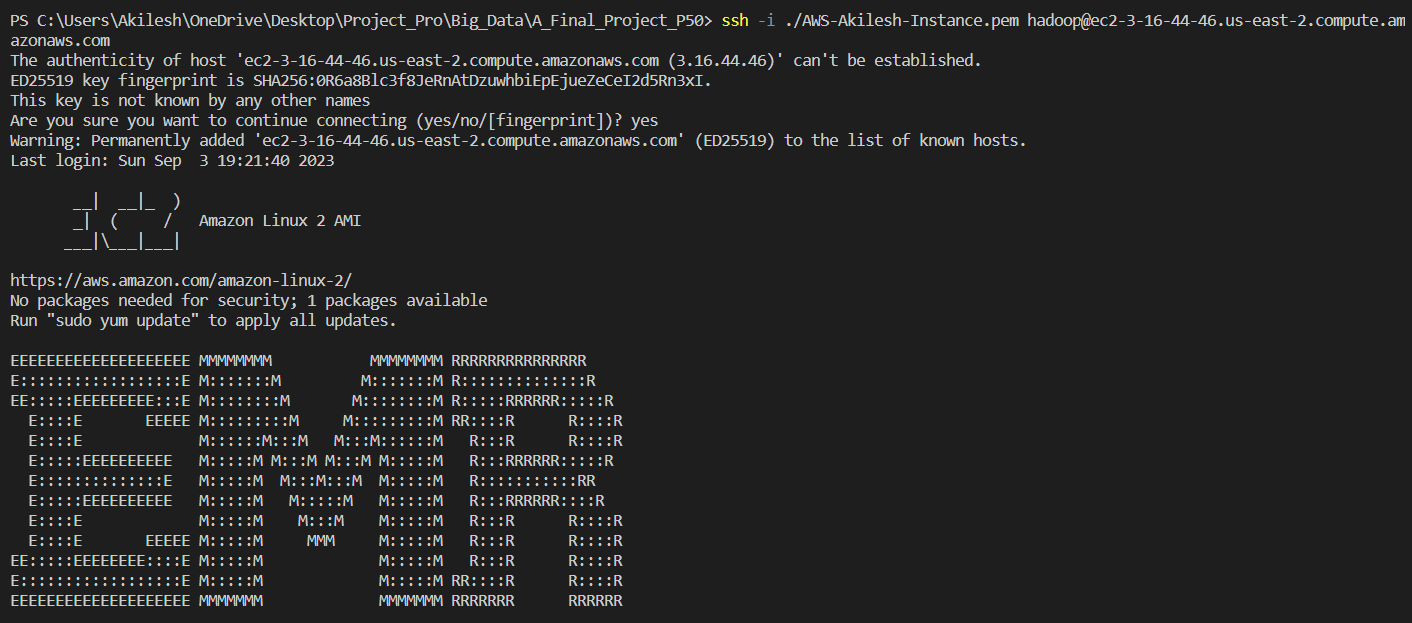


**Enabling SSH connection from local for Maste node**

****

**SSH connection to cluster established:**

ssh -i ./AWS-Akilesh-Instance.pem hadoop@ec2-3-16-44-46.us-east-2.compute.amazonaws.com

****

**Now Pasting the Pyspark code in VI editor:**

from pyspark.sql import SparkSession

S3\_DATA\_INPUT\_PATH = "s3://p50-bigdata-proto-filtered/unsw-nb15/proto\_filtered/run-1693765110601-part-r-00000"

S3\_DATA\_OUTPUT\_PATH = "s3://p50-bigdata-filtered-attack-records/unsw-nb15/p50\_attack\_records"

def main():

    spark = SparkSession.builder.appName('p50\_projectProDemo').getOrCreate()

    # Read CSV dataset with header

    df = spark.read.option("header", "true").csv(S3\_DATA\_INPUT\_PATH)

    # Display the schema of the DataFrame

    df.printSchema()

    # Filter records where Label equals 1

    filtered\_df = df.filter(df.Label == '1')

    print(f'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*The total number of records in the input data set is {df.count()}\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

    print(f'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*The total number of records in the filtered data set is {filtered\_df.count()}\*\*\*\*\*\*\*\*\*\*\*\*')

    filtered\_df.show(10)  # Display the first 10 records of the filtered DataFrame

    # Write the filtered DataFrame to S3 in CSV format

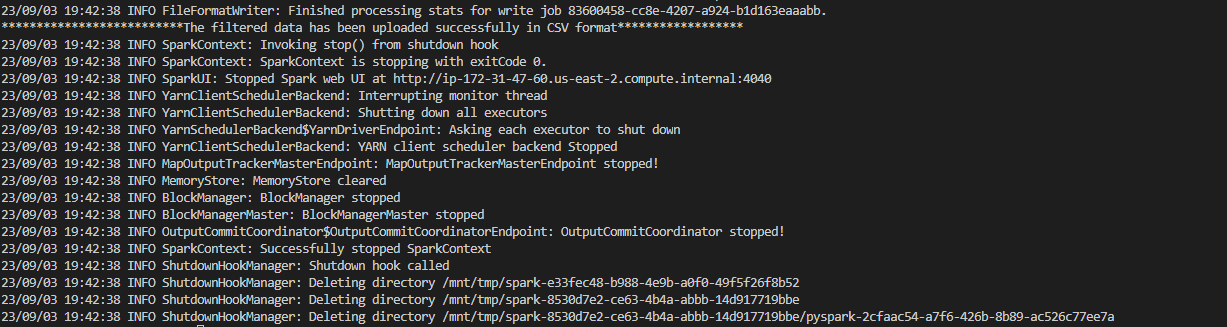
    filtered\_df.write.mode('overwrite').option("header", "true").csv(S3\_DATA\_OUTPUT\_PATH)

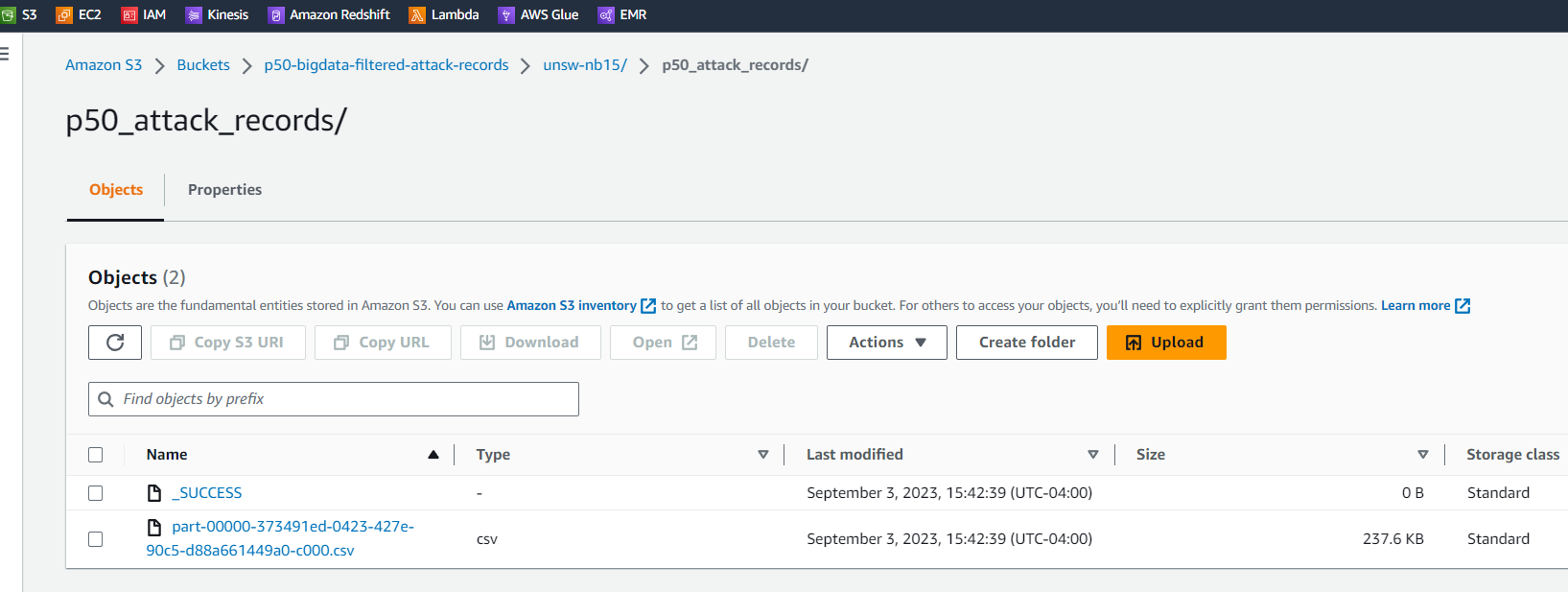
    print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*The filtered data has been uploaded successfully in CSV format\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**Successfully running the Pyspark script and confirming the data load to output S3 path**

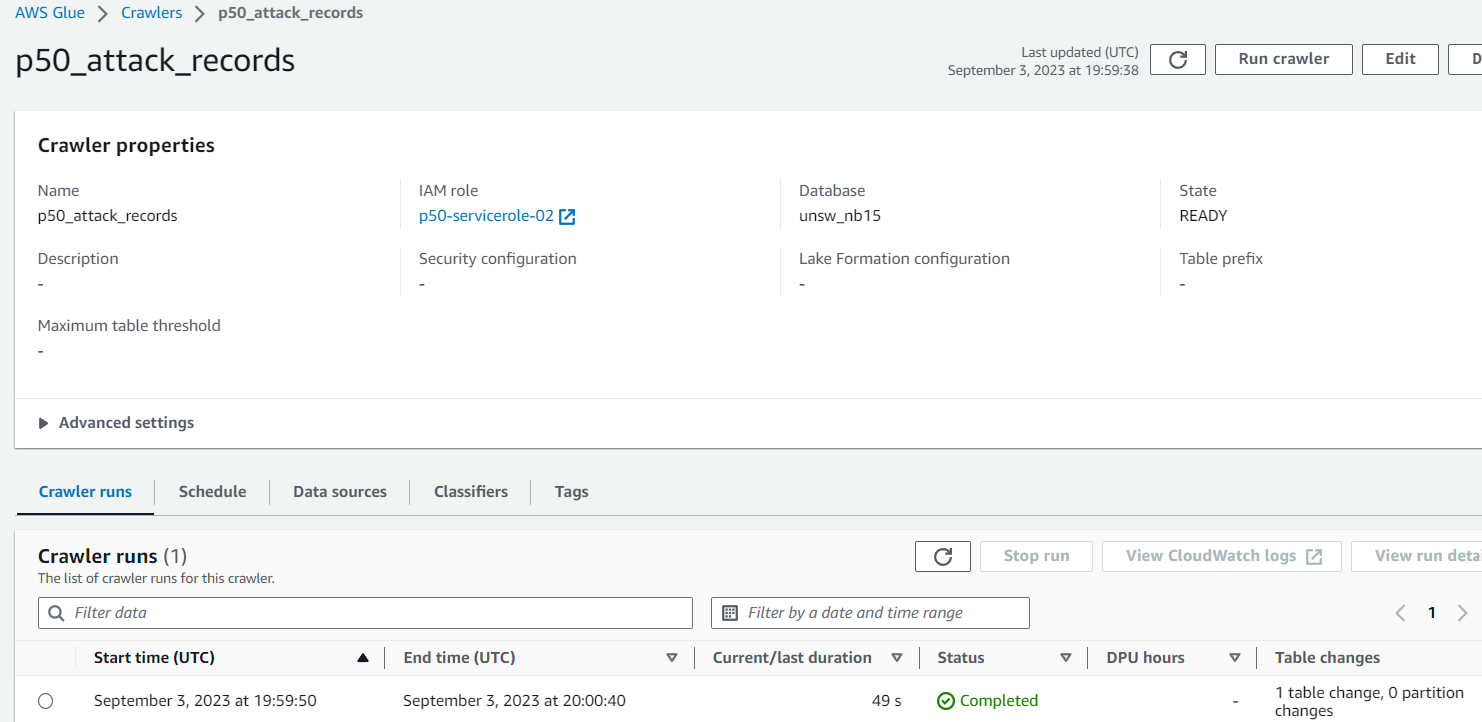




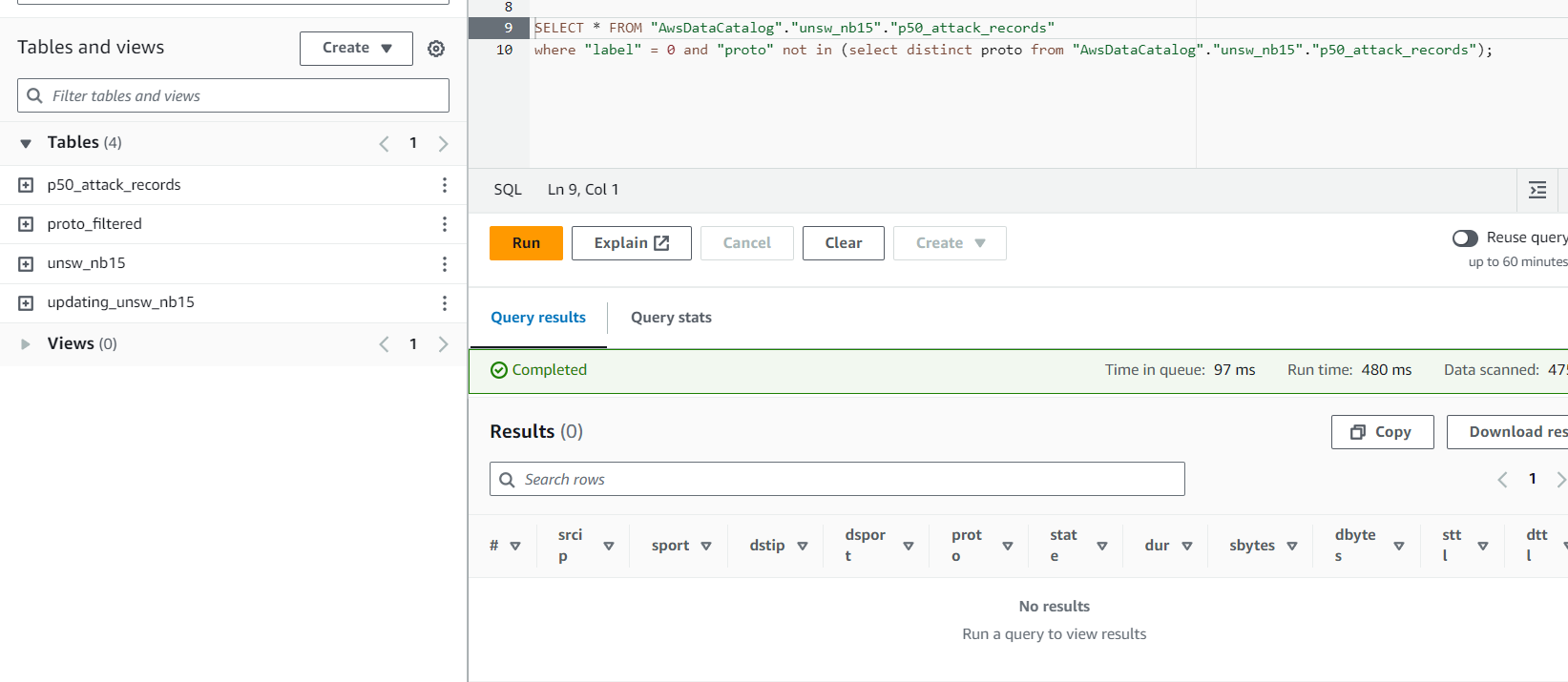
### (1.8) Validation: Attack Records

We will validate this by creating a Crawler and query this data using Athena. Ensure that you get no records where "label" is 0, and "proto" not in ("tcp", "udp")

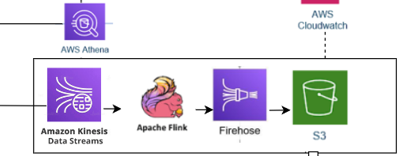
* Crawler created and Run



Now lets query in **Athena** and verify the results:

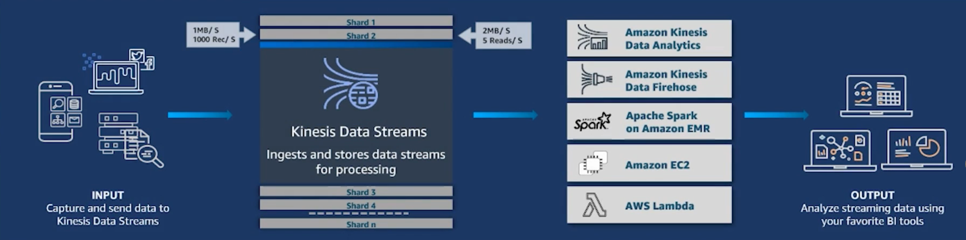


## Phase: 2

****

**Knowledge for this pipeline:**

**Kinesis Data Streams**



Amazon Kinesis Data Streams is a managed service provided by AWS for ingesting, processing, and analyzing real-time streaming data. It allows you to collect and process large streams of data records in real-time, enabling you to build applications that require processing of high volumes of streaming data such as logs, metrics, financial transactions, social media feeds, and more.

**Apache Flink**

****

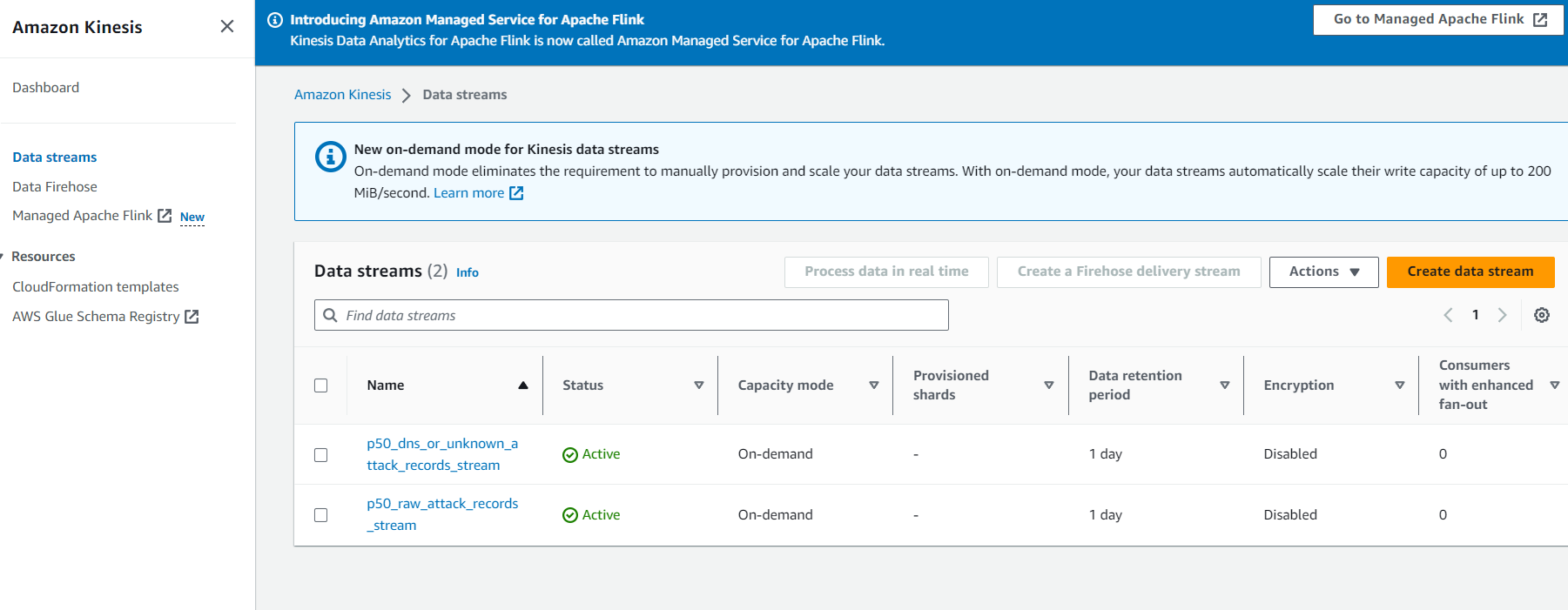
Apache Flink is an open-source stream processing framework designed to handle large-scale, real-time data processing and analytics applications. It provides a powerful and flexible platform for processing continuous data streams, as well as batch data processing.

**Kinesis Firehose**

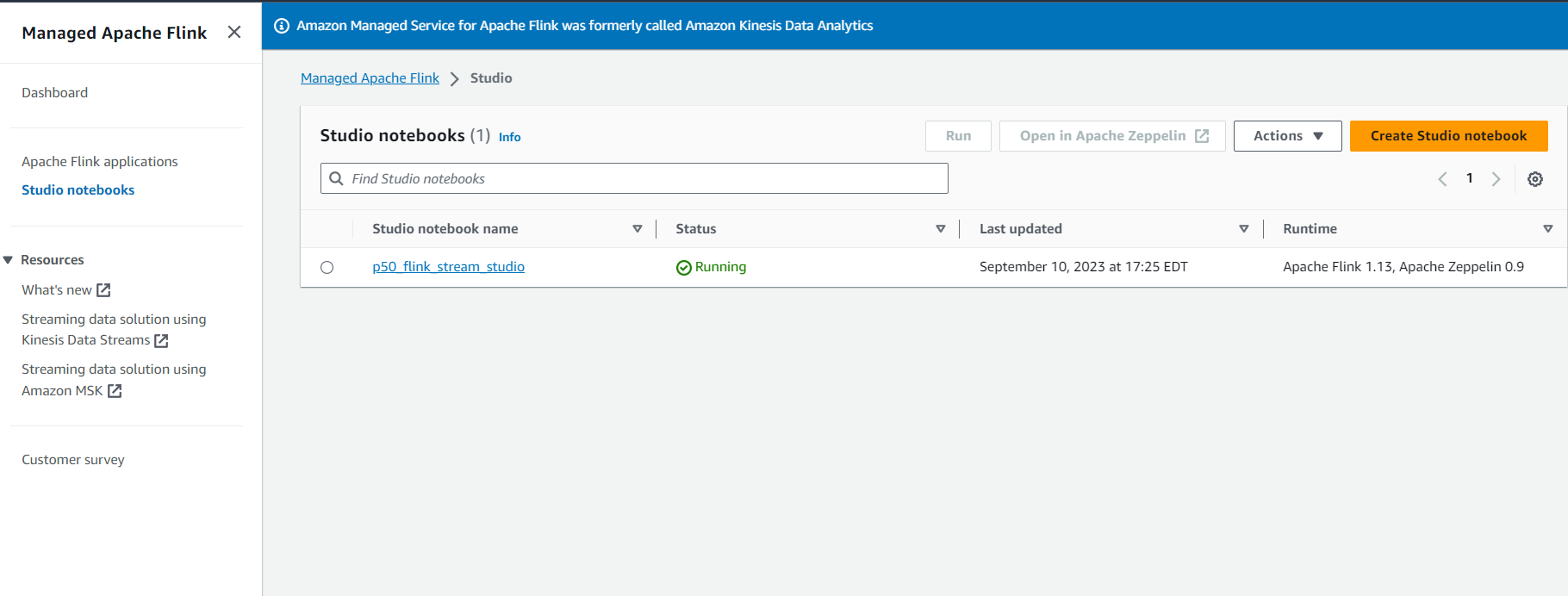
****

Amazon Kinesis Data Firehose is a fully managed service provided by AWS for reliably ingesting, transforming, and loading large volumes of streaming data into data lakes, data stores, and analytics services. It simplifies the process of ingesting and preparing streaming data for storage and analysis.

### (2.1) Create Data Streams



### (2.2) Creation of Studio Notebook for Apache Flink



**Flink Code (Zeplin Notebook)**

%flink.ssql

/\*Option 'IF NOT EXISTS' can be used, to protect the existing Schema \*/

DROP TABLE IF EXISTS p50\_stream1\_records\_table;

CREATE TABLE p50\_stream1\_records\_table (

  `srcip` VARCHAR(50),

  `sport` bigint,

  `dstip` VARCHAR(50),

  `dsport` bigint,

  `proto` VARCHAR(50),

  `state` VARCHAR(50),

  `dur` VARCHAR(50),

  `sbytes` bigint,

  `dbytes` bigint,

  `sttl` bigint,

  `dttl` bigint,

  `sloss` bigint,

  `dloss` bigint,

  `service` VARCHAR(50),

  `sload` VARCHAR(50),

  `dload` double,

  `spkts` bigint,

  `dpkts` bigint,

  `swin` bigint,

  `dwin` bigint,

  `stcpb` bigint,

  `dtcpb` bigint,

  `smeansz` bigint,

  `dmeansz` bigint,

  `trans\_depth` bigint,

  `res\_bdy\_len` bigint,

  `sjit` double,

  `djit` double,

  `stime` bigint,

  `ltime` bigint,

  `sintpkt` double,

  `dintpkt` double,

  `tcprtt` double,

  `synack` double,

  `ackdat` double,

  `is\_sm\_ips\_ports` bigint,

  `ct\_state\_ttl` bigint,

  `ct\_flw\_http\_mthd` bigint,

  `is\_ftp\_login` bigint,

  `ct\_ftp\_cmd` bigint,

  `ct\_srv\_src` bigint,

  `ct\_srv\_dst` bigint,

  `ct\_dst\_ltm` bigint,

  `ct\_src\_ ltm` bigint,

  `ct\_src\_dport\_ltm` bigint,

  `ct\_dst\_sport\_ltm` bigint,

  `ct\_dst\_src\_ltm` bigint,

  `attack\_cat` VARCHAR(50),

  `label` bigint,

  `Txn\_Timestamp` TIMESTAMP(3)

)

PARTITIONED BY (service)

WITH (

  'connector' = 'kinesis',

  'stream' = 'p50\_raw\_attack\_records\_stream',

  'aws.region' = 'us-east-2',

  'scan.stream.initpos' = 'LATEST',

  'format' = 'json',

  'json.timestamp-format.standard' = 'ISO-8601'

);

/\*Option 'IF NOT EXISTS' can be used, to protect the existing Schema \*/

DROP TABLE IF EXISTS p50\_stream1\_records\_table\_results;

CREATE TABLE p50\_stream1\_records\_table\_results (

  `srcip` VARCHAR(50),

  `sport` bigint,

  `dstip` VARCHAR(50),

  `dsport` bigint,

  `proto` VARCHAR(50),

  `state` VARCHAR(50),

  `dur` VARCHAR(50),

  `sbytes` bigint,

  `dbytes` bigint,

  `sttl` bigint,

  `dttl` bigint,

  `sloss` bigint,

  `dloss` bigint,

  `service` VARCHAR(50),

  `sload` VARCHAR(50),

  `dload` double,

  `spkts` bigint,

  `dpkts` bigint,

  `swin` bigint,

  `dwin` bigint,

  `stcpb` bigint,

  `dtcpb` bigint,

  `smeansz` bigint,

  `dmeansz` bigint,

  `trans\_depth` bigint,

  `res\_bdy\_len` bigint,

  `sjit` double,

  `djit` double,

  `stime` bigint,

  `ltime` bigint,

  `sintpkt` double,

  `dintpkt` double,

  `tcprtt` double,

  `synack` double,

  `ackdat` double,

  `is\_sm\_ips\_ports` bigint,

  `ct\_state\_ttl` bigint,

  `ct\_flw\_http\_mthd` bigint,

  `is\_ftp\_login` bigint,

  `ct\_ftp\_cmd` bigint,

  `ct\_srv\_src` bigint,

  `ct\_srv\_dst` bigint,

  `ct\_dst\_ltm` bigint,

  `ct\_src\_ ltm` bigint,

  `ct\_src\_dport\_ltm` bigint,

  `ct\_dst\_sport\_ltm` bigint,

  `ct\_dst\_src\_ltm` bigint,

  `attack\_cat` VARCHAR(50),

  `label` bigint,

  `Txn\_Timestamp` TIMESTAMP(3)

)

PARTITIONED BY (service)

WITH (

  'connector' = 'kinesis',

  'stream' = 'p50\_dns\_or\_unknown\_attack\_records\_stream',

  'aws.region' = 'us-east-2',

  'format' = 'json',

  'json.timestamp-format.standard' = 'ISO-8601'

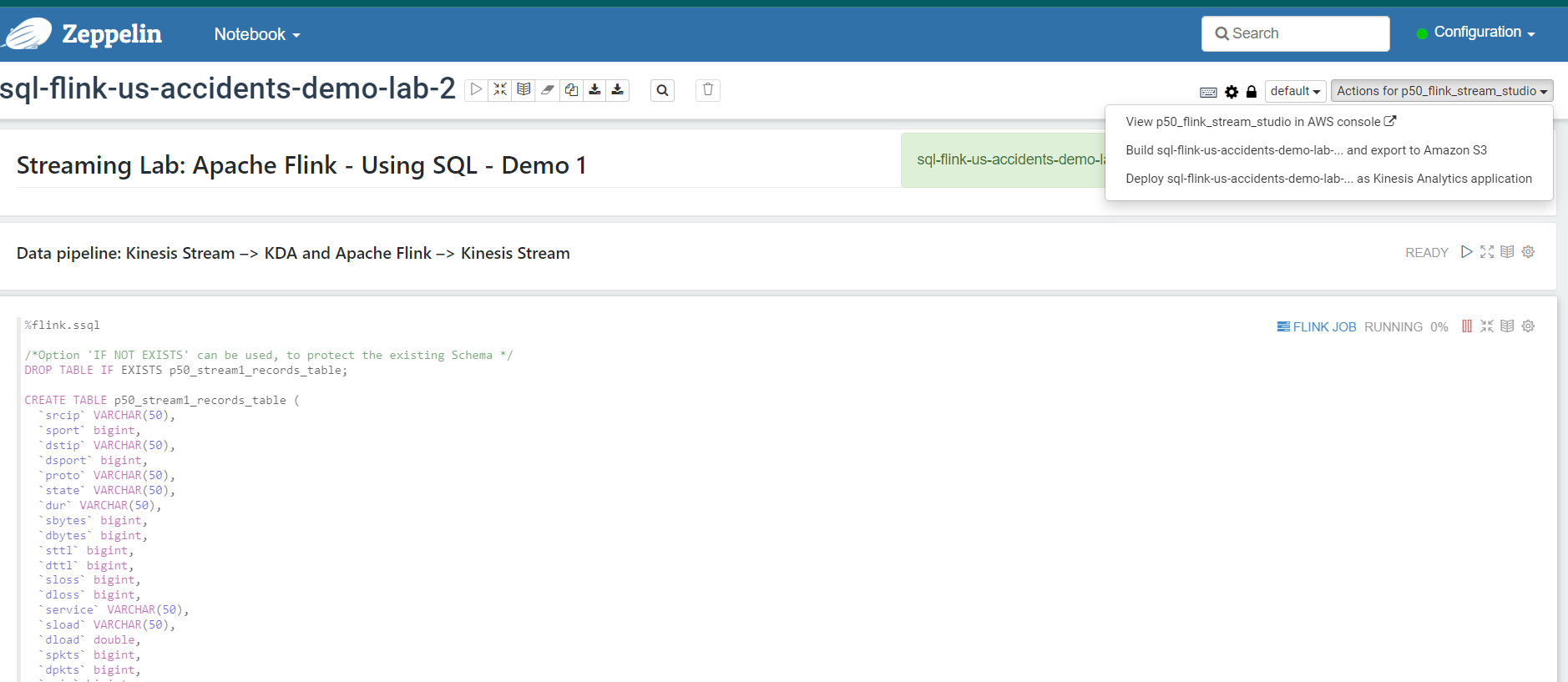
);

insert into p50\_stream1\_records\_table\_results

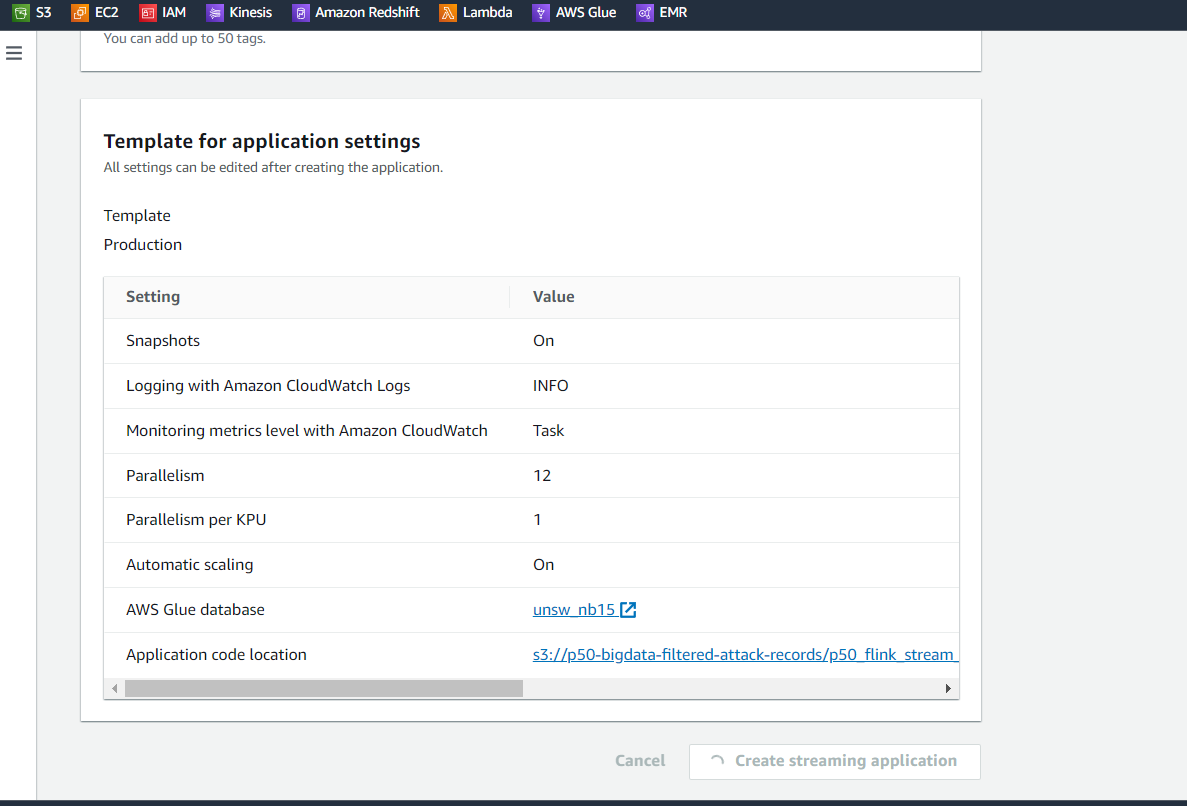
select  \*

from p50\_stream1\_records\_table where service in ('-', 'dns');

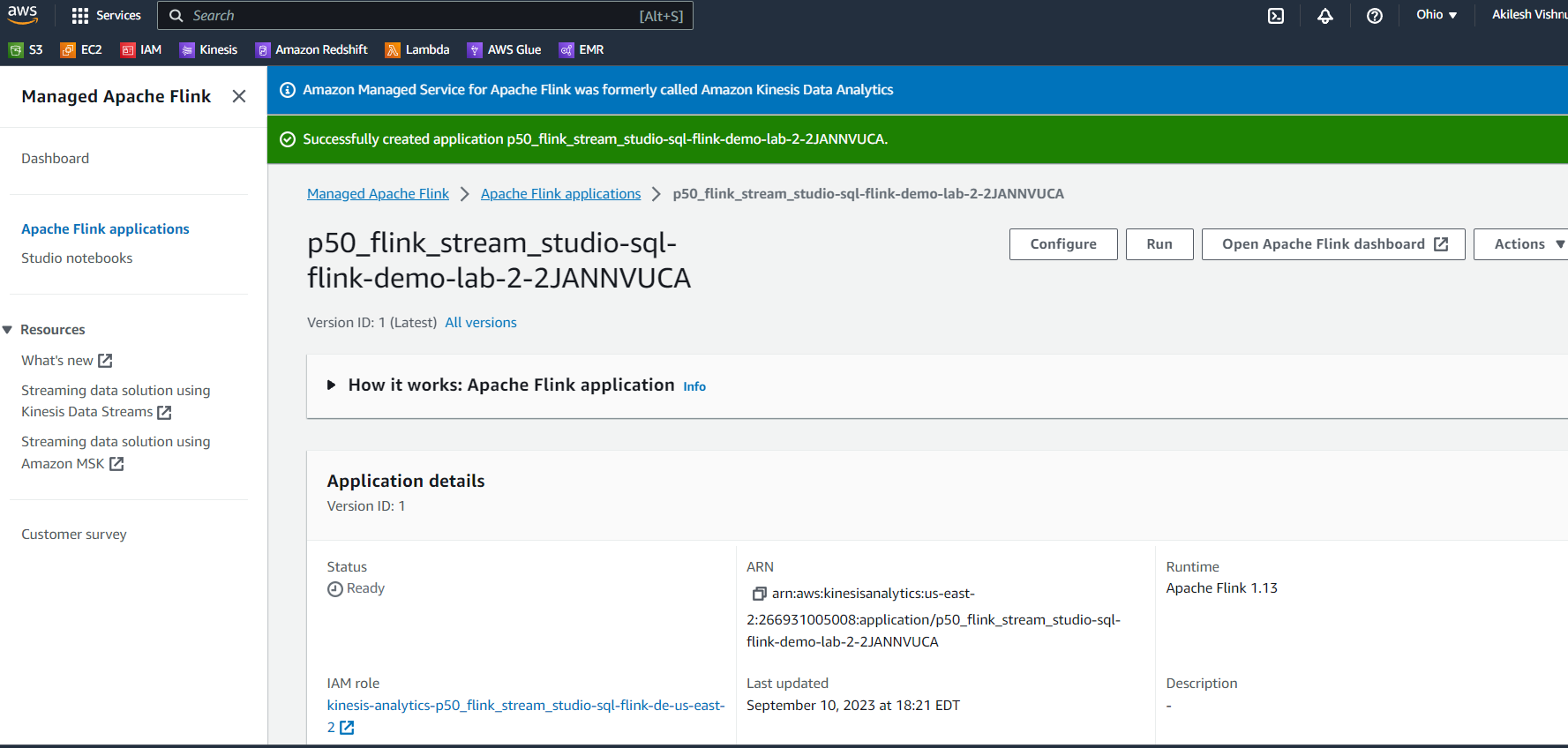
### (2.3) Creating Streaming Application



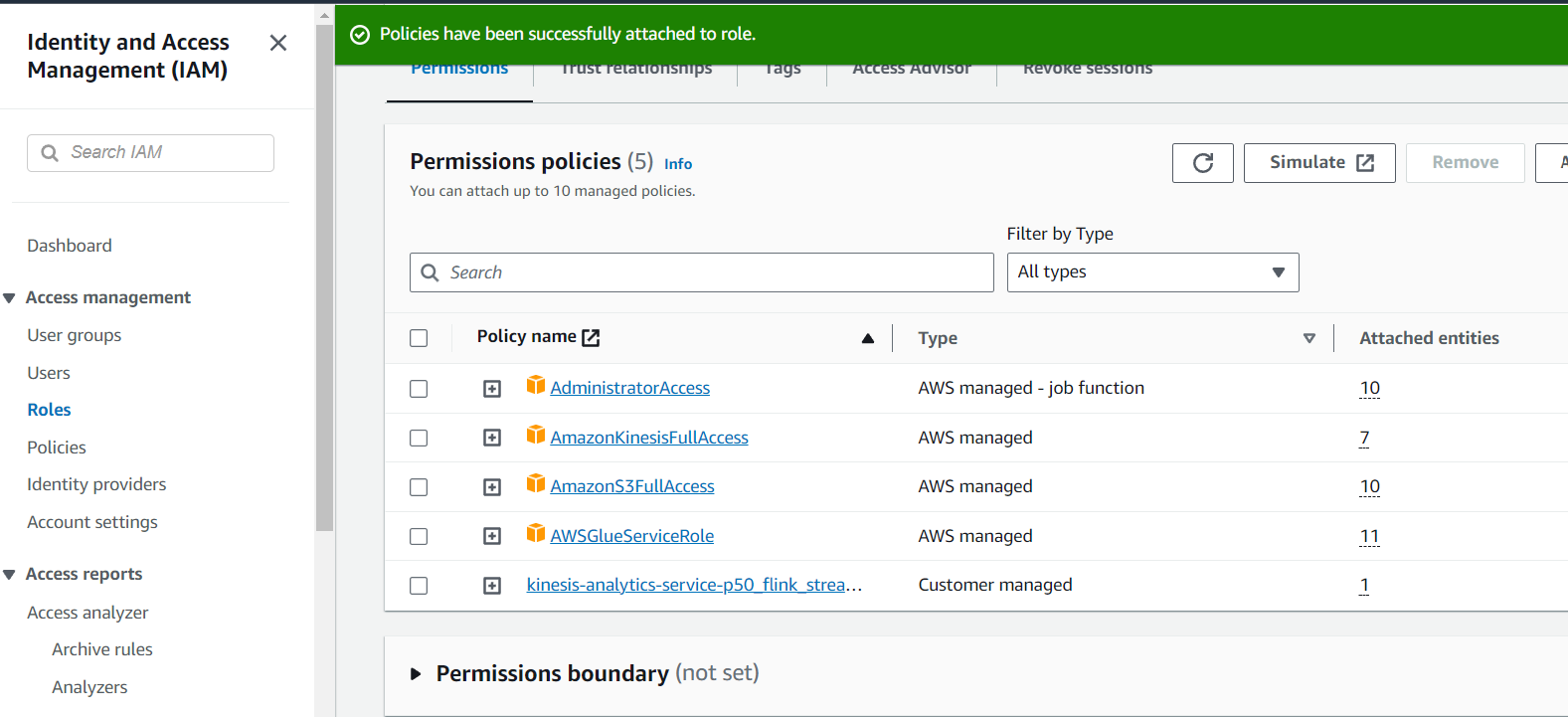




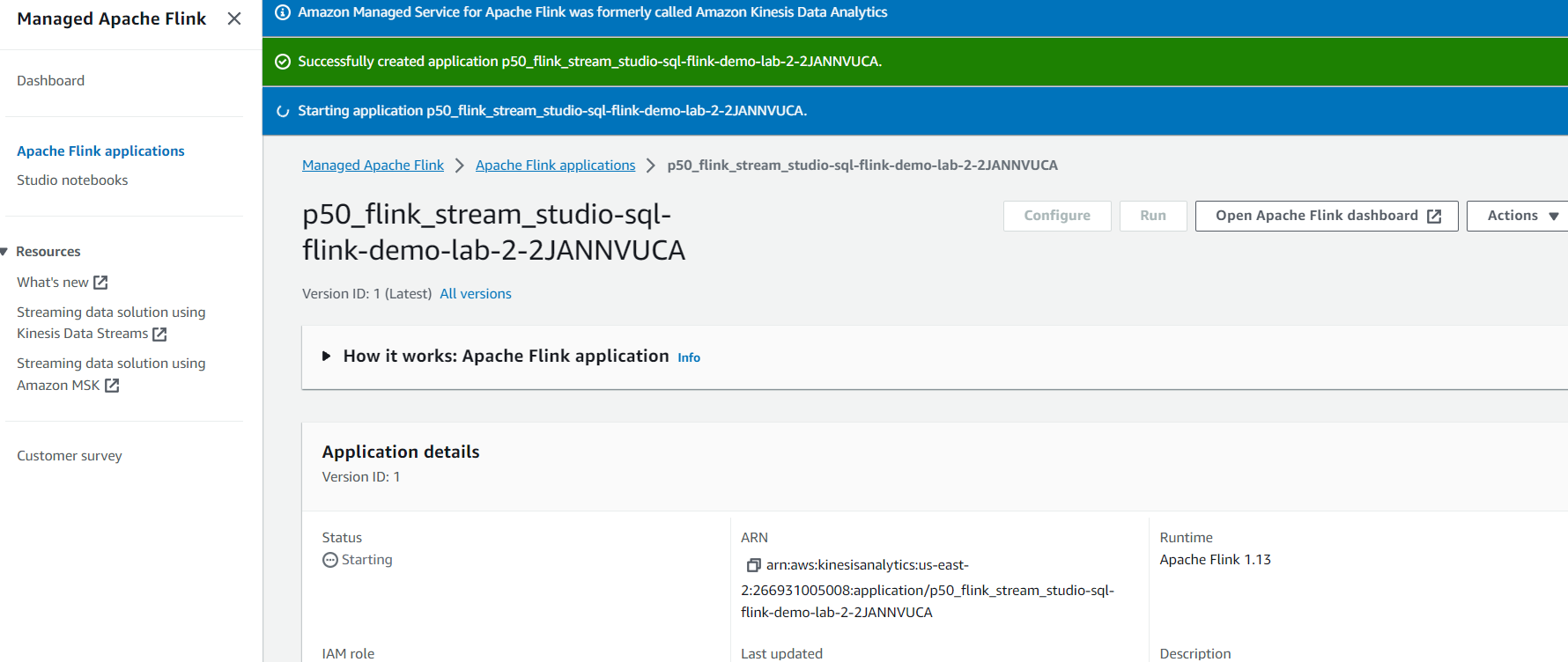
**Streaming Application successfully created**



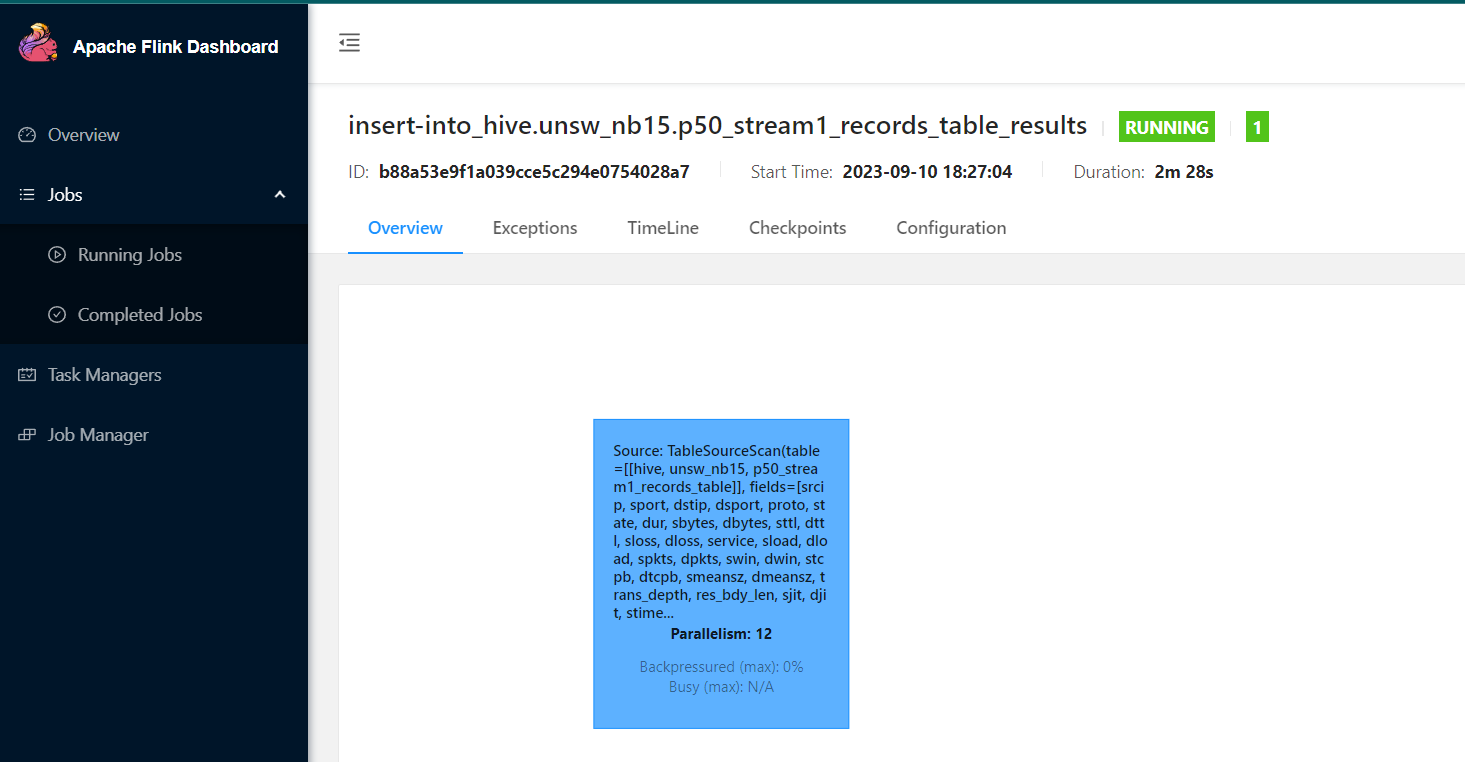
**Check and add necessary roles**



**Run the application:**



**Verify if the Flink is running properly**



**Validate if the Application is running fine by triggering the Stream 1**

import boto3

import csv

import json

import dateutil.parser as parser

from time import sleep

from datetime import datetime

# AWS SDK Local Configuration

aws\_access\_key\_id = "123"

aws\_secret\_access\_key = "1234"

region\_name = "us-east-2"

# Create S3 Client with Custom Credentials

s3 = boto3.client('s3', region\_name=region\_name,

                  aws\_access\_key\_id=aws\_access\_key\_id,

                  aws\_secret\_access\_key=aws\_secret\_access\_key)

# Create S3 Resource with Custom Credentials

s3\_resource = boto3.resource('s3', region\_name=region\_name,

                              aws\_access\_key\_id=aws\_access\_key\_id,

                              aws\_secret\_access\_key=aws\_secret\_access\_key)

# Create Kinesis Client with Custom Credentials

kinesis\_client = boto3.client('kinesis', region\_name=region\_name,

                              aws\_access\_key\_id=aws\_access\_key\_id,

                              aws\_secret\_access\_key=aws\_secret\_access\_key)

# Env. variables; i.e. can be OS variables in Lambda

kinesis\_stream\_name = 'p50\_raw\_attack\_records\_stream'

#Partition key not needed as dataset is small

streaming\_partition\_key = 'service'

# Function can be converted to Lambda;

#   i.e. by iterating the S3-put events records; e.g. record['s3']['bucket']['name']

def stream\_data\_simulator(input\_s3\_bucket, input\_s3\_key):

  s3\_bucket = input\_s3\_bucket

  s3\_key = input\_s3\_key

  # Read CSV Lines and split the file into lines

  csv\_file = s3\_resource.Object(s3\_bucket, s3\_key)

  s3\_response = csv\_file.get()

  lines = s3\_response['Body'].read().decode('utf-8').split('\n')

  for row in csv.DictReader(lines):

      try:

          # Convert to JSON, to make it easier to work in Kinesis Analytics

          line\_json = json.dumps(row)

          json\_load = json.loads(line\_json)

          # Adding fake txn ts:

          json\_load['Txn\_Timestamp'] = datetime.now().isoformat()

          # Write to Kinesis Streams:

          response = kinesis\_client.put\_record(StreamName=kinesis\_stream\_name,Data=json.dumps(json\_load, indent=4),PartitionKey=str(json\_load[streaming\_partition\_key]))

          print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*",response)

          # Adding a temporary pause, for demo-purposes:

          sleep(0.250)

      except Exception as e:

          print('Error: {}'.format(e))

          print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

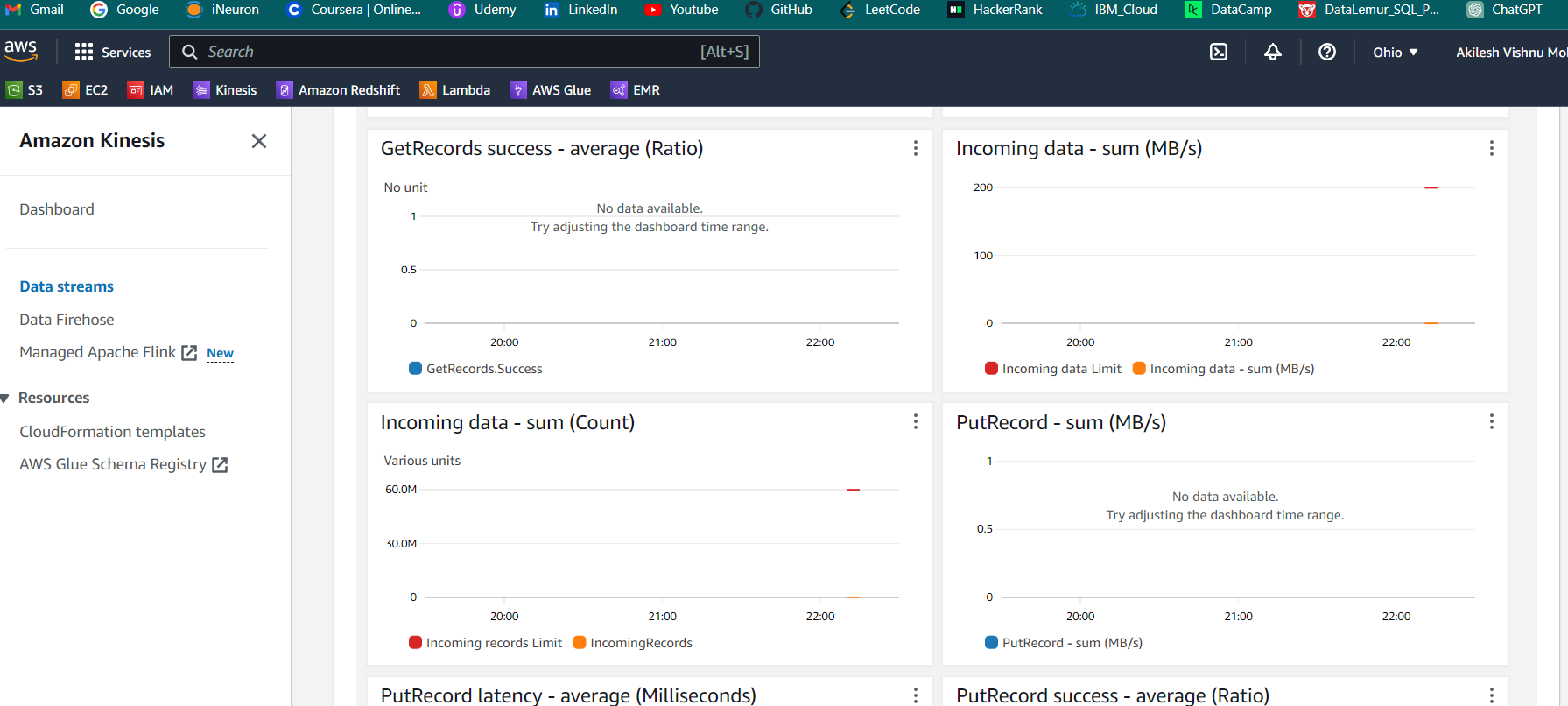
# Run stream:

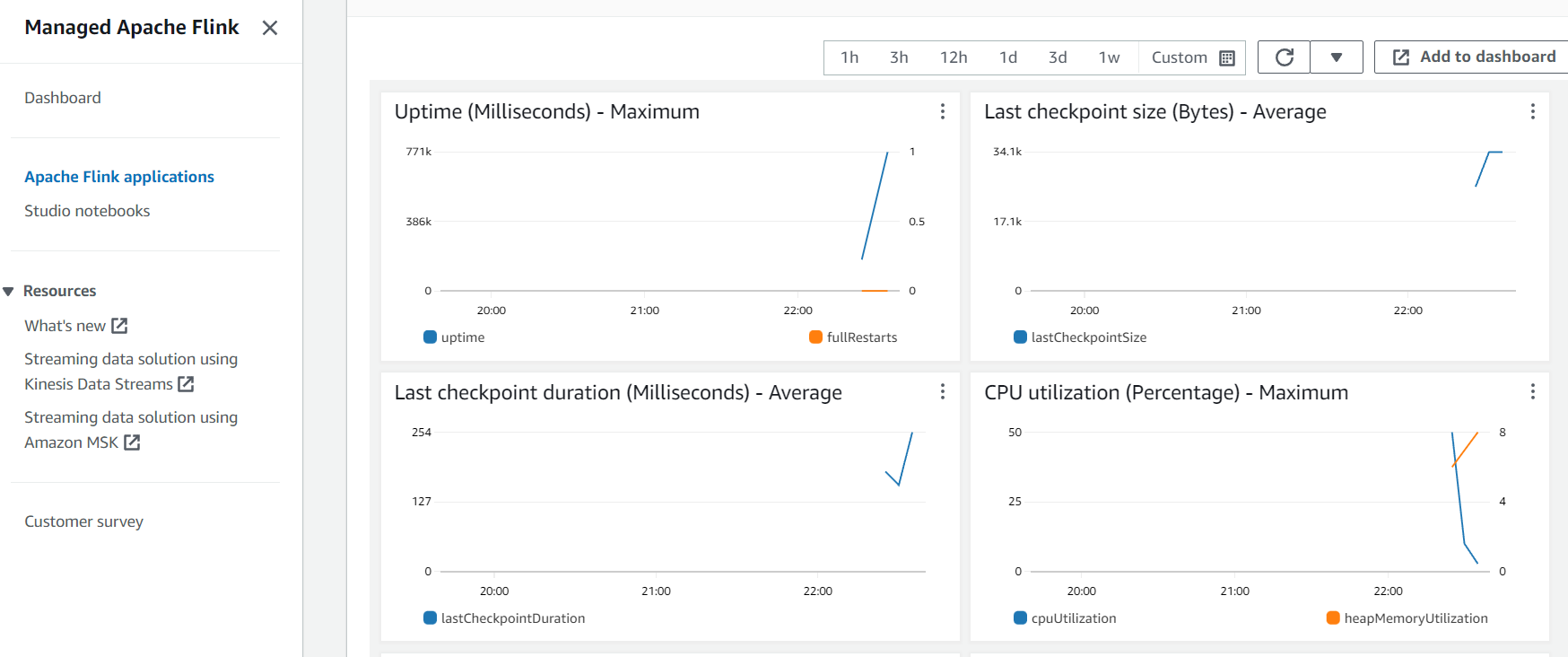
for i in range(0, 3):

  stream\_data\_simulator(input\_s3\_bucket="p50-bigdata-filtered-attack-records", input\_s3\_key="unsw-nb15/p50\_attack\_records/part-00000-373491ed-0423-427e-90c5-d88a661449a0-c000.csv")

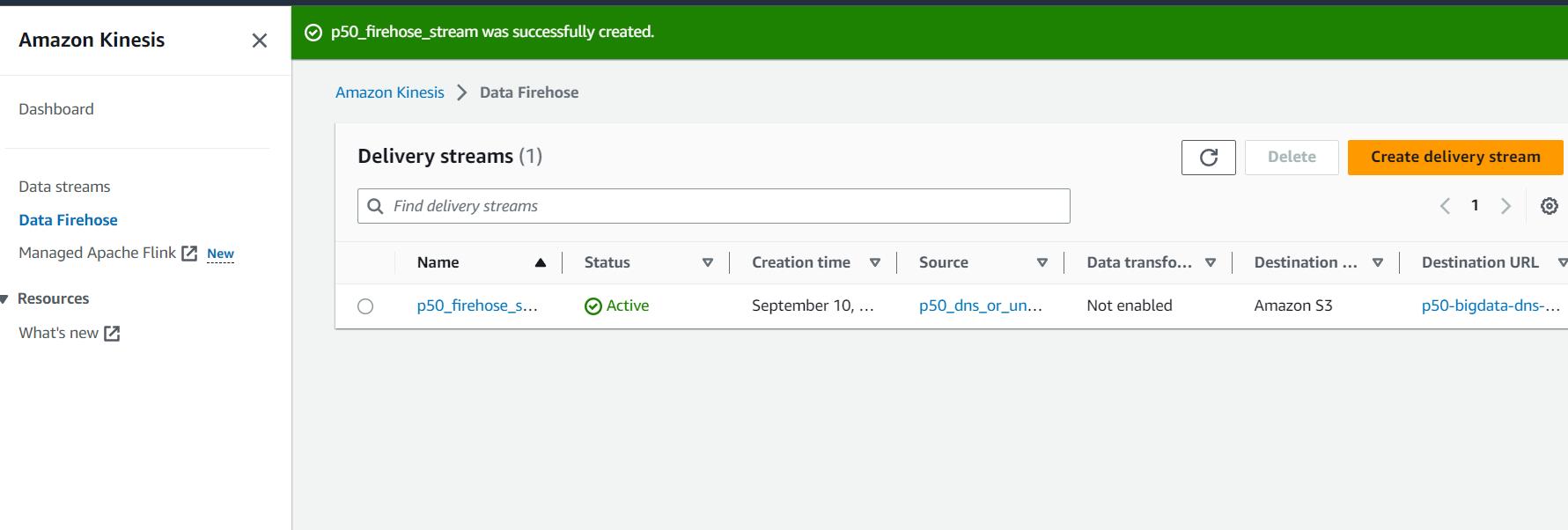
****

**You can see incoming data in Stream 2: p50\_dns\_or\_unknown\_attack\_records\_stream**

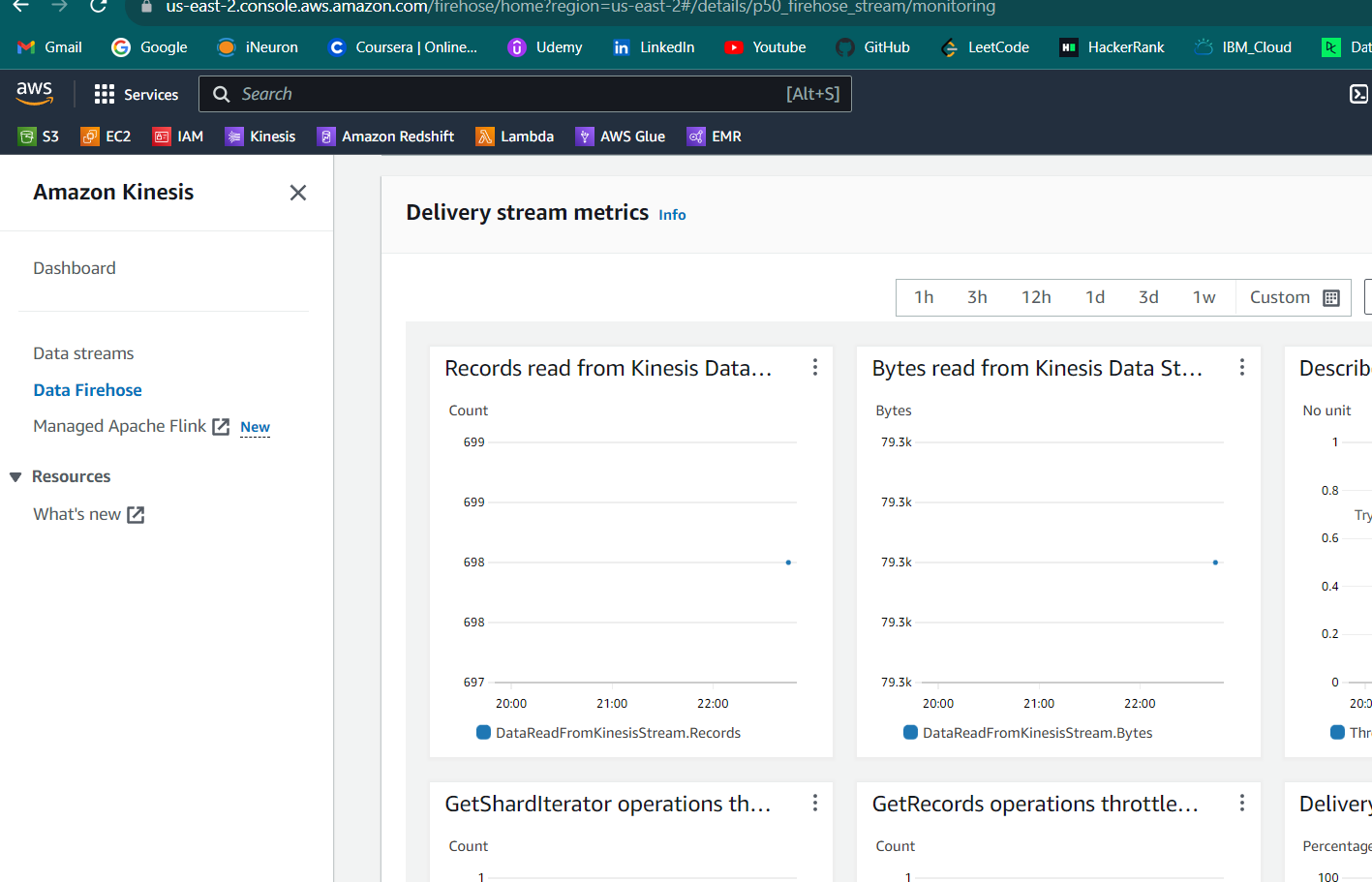




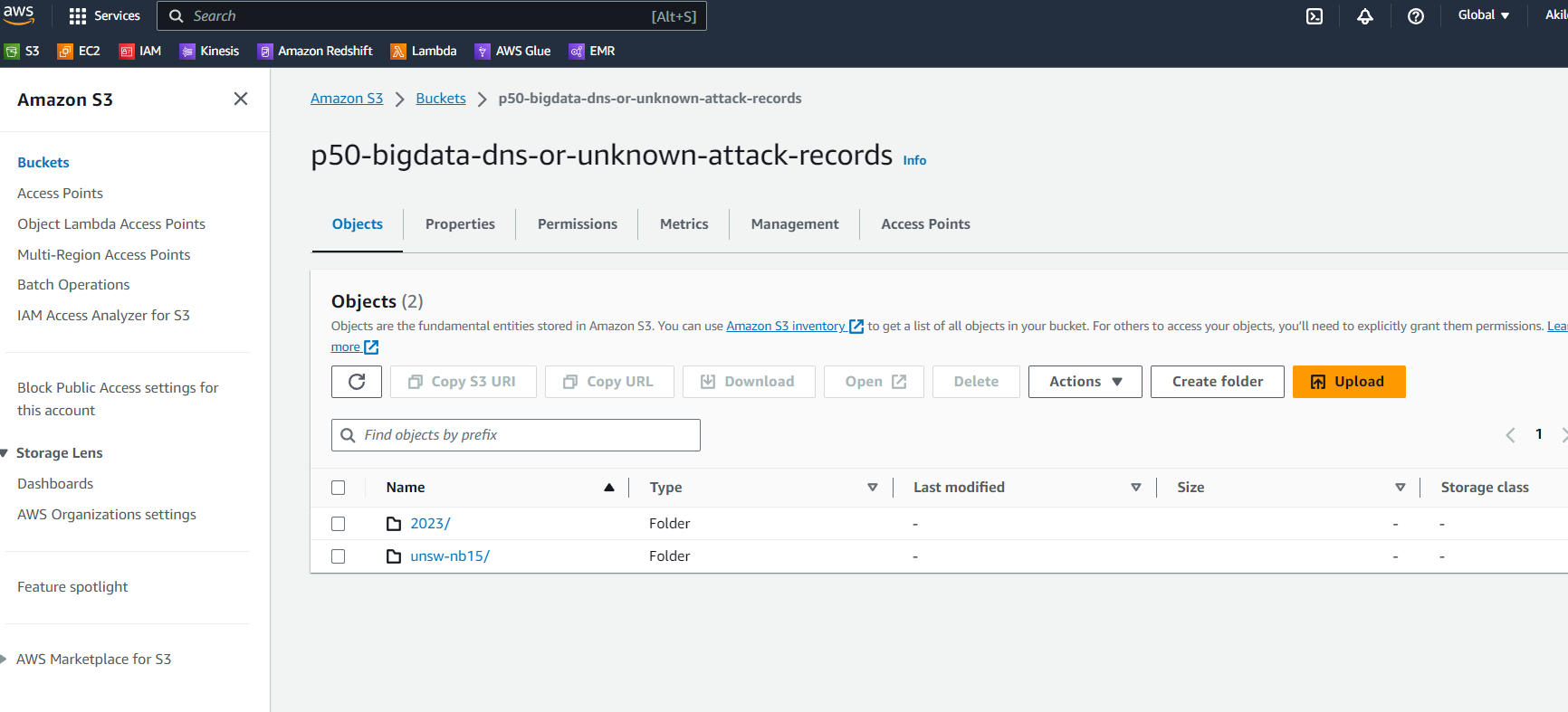
### (2.4) Creating Firehose Delivery Stream:



We can see Data coming to the Delivery stream through Kinesis Firehose



We can validate this now in the S3 bucket that the data is coming in

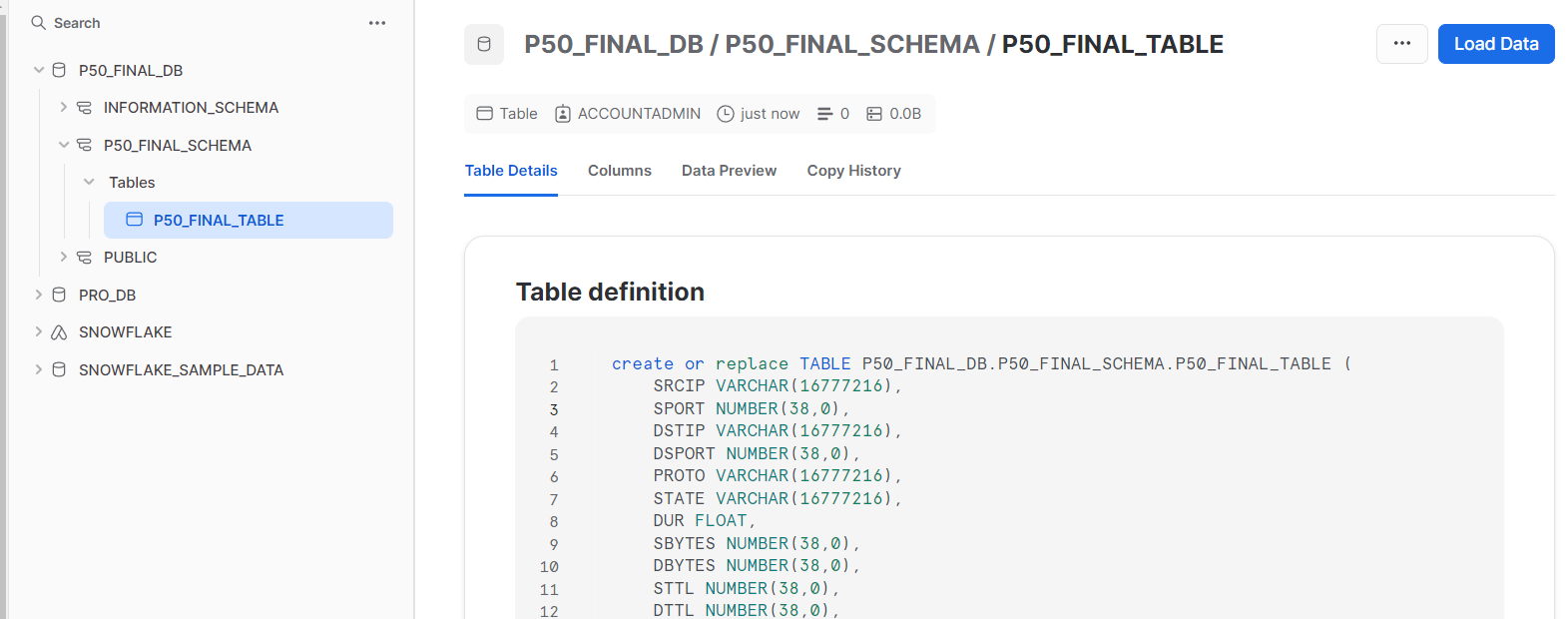


## Phase: 3



### (3.1) Setting up Snowflake

So first we will setup the Snowflake Environment for the execution of the project. So first we will create a snowflake account followed by creation of the below **Database, Schema and Table** created with the below definitions, needed for the execution of the project.



### (3.2) Setting connection from AWS to Snowflake

Now we need to securely establish connection for interaction between AWS S3 and Snowflake. We cant expose out AWS access key credentials as they pose a risk to security so we will be creating proper IAM roles and creating Storage that will be used to establish connection

**IAM Role Creation:**

So we are creating a managed IAM policy for only this

bucket: **p50-bigdata-dns-or-unknown-attack-records**

For this we will be creating a policy and a for accessing the policy

Policy: **P50\_SNOWFLAKE\_ACCESS\_POLICY**

We are creating the policy for establishing connection from S3 to snowflake and reading the content

{

    "Version": "2012-10-17",

    "Statement": [

        {

            "Effect": "Allow",

            "Action": [

                "s3:PutObject",

                "s3:GetObject",

                "s3:GetObjectVersion",

                "s3:DeleteObject",

                "s3:DeleteObjectVersion"

            ],

            "Resource": "arn:aws:s3:::p50-bigdata-dns-or-unknown-attack-records/unsw-nb15/\*"

        },

        {

            "Effect": "Allow",

            "Action": "s3:ListBucket",

            "Resource": "arn:aws:s3:::p50-bigdata-dns-or-unknown-attack-records",

            "Condition": {

                "StringLike": {

                    "s3:prefix": [

                        "Input/\*"

                    ]

                }

            }

        }

    ]

}



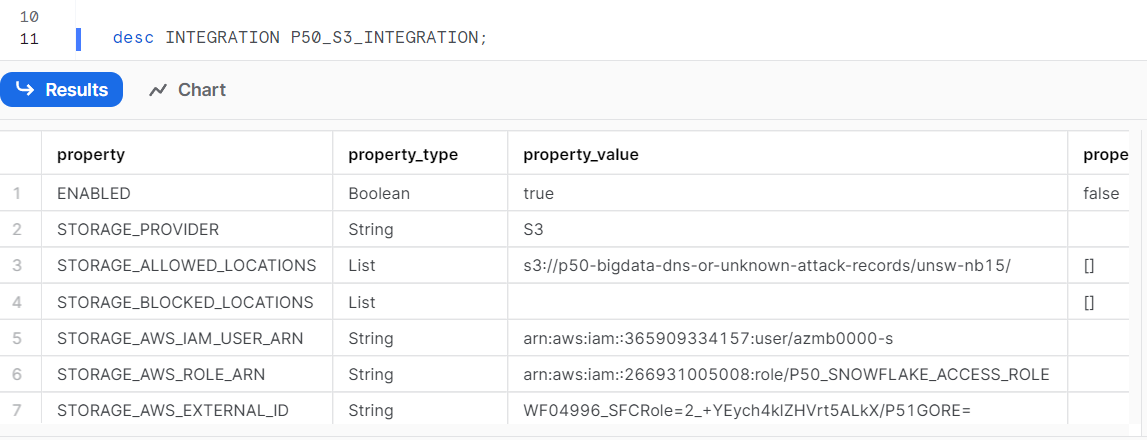
Role: **P50\_SNOWFLAKE\_ACCESS\_ROLE**

arn:aws:iam::266931005008:role/P50\_SNOWFLAKE\_ACCESS\_ROLE

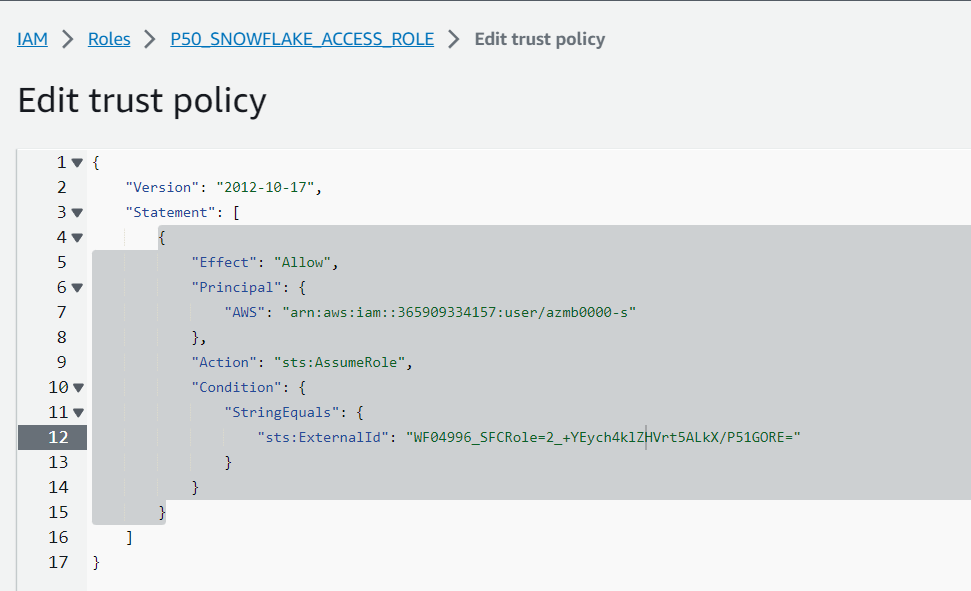


### (3.3) Snowflake Storage Integration

****

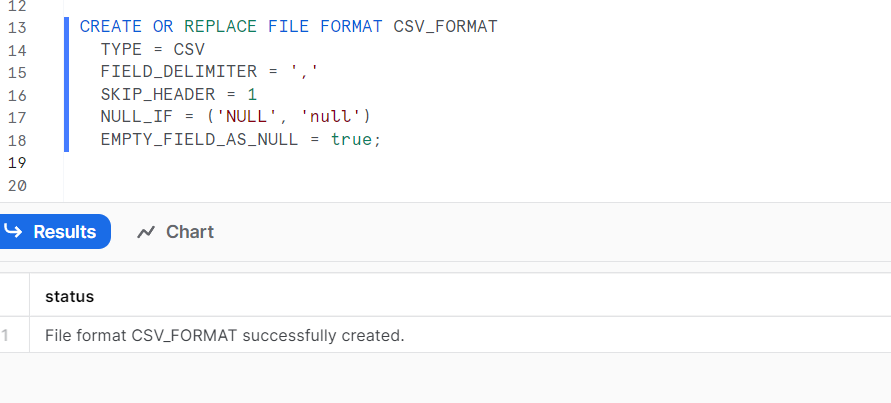
****

Now we can see the IAM\_USER\_ARN and AWS\_EXTERNAL\_ID that Snowflake has created for us (which is the external ARN that AWS needs to recognize Snowflake) and we will have to update this in the managed Role.

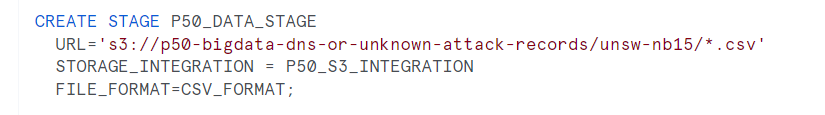
****

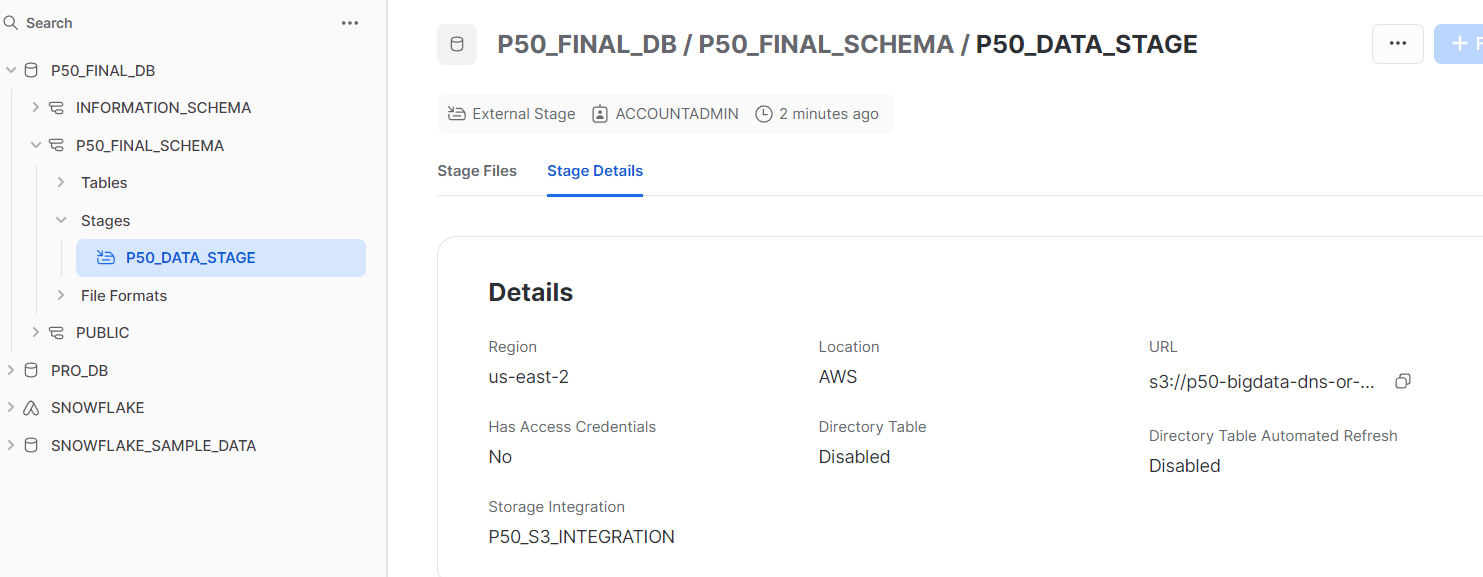
### (3.4) Set up snowflake for loading the data from S3

1. **Fileformat creation**

****

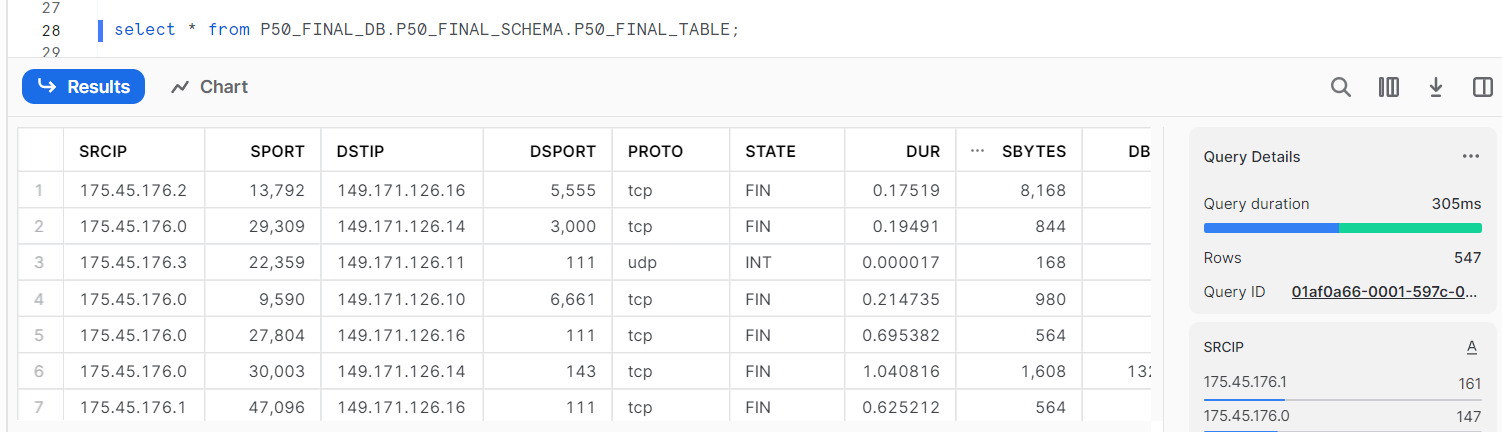
1. Stage Creation:





1. Loading Data from Stage





### (3.5) Airflow Orchestration

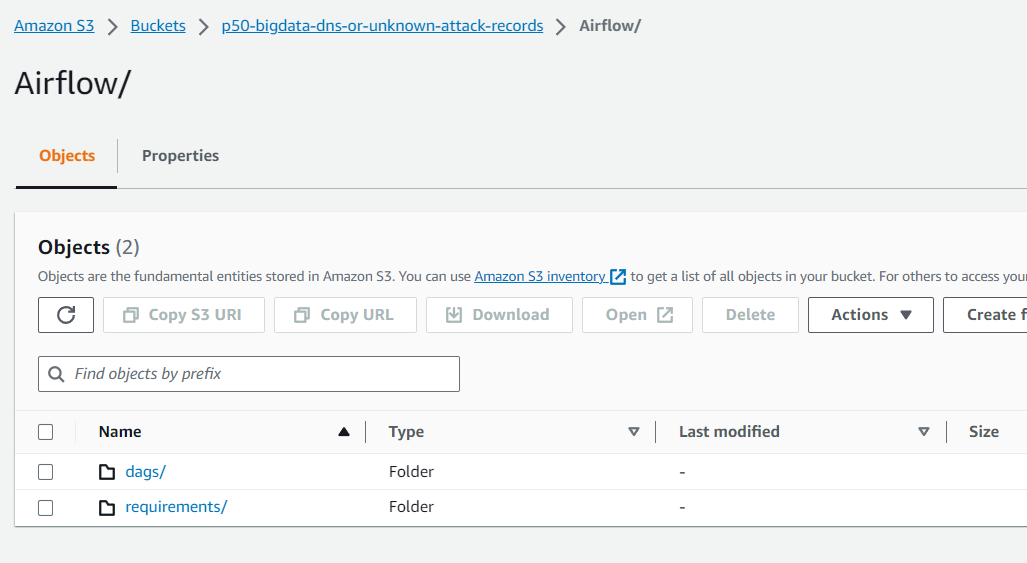
So far we have done all the loading in snowflake manually and this is not a efficient way so now we are going to use Airflow for orchestrating the data flow.

So we can use Airflow through docker but here I am using **Amazon Managed Apache Airflow**

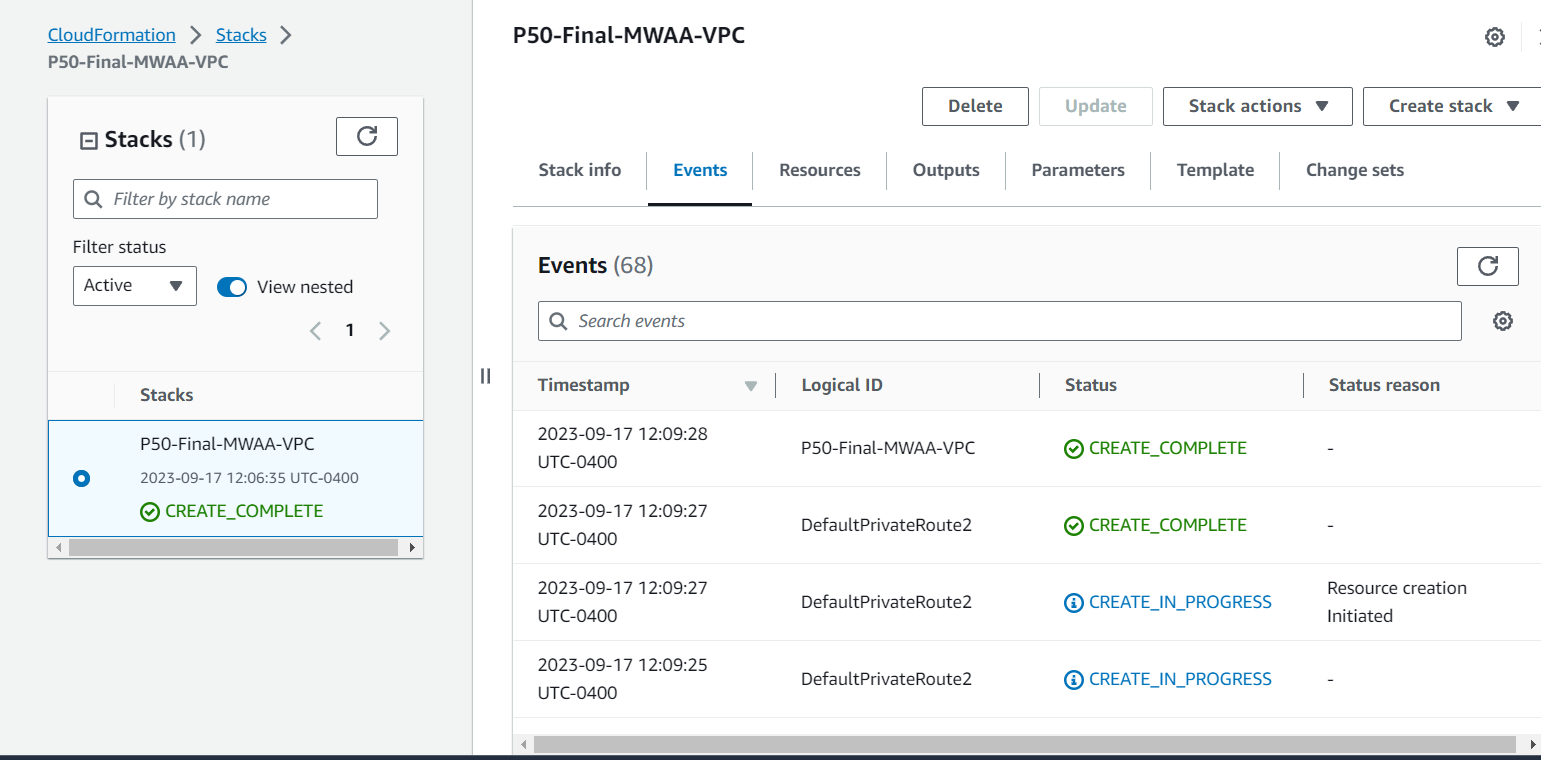
This is a costly service so be aware of the cost it comes along with it based on that configure the nodes and processing units needed for the task

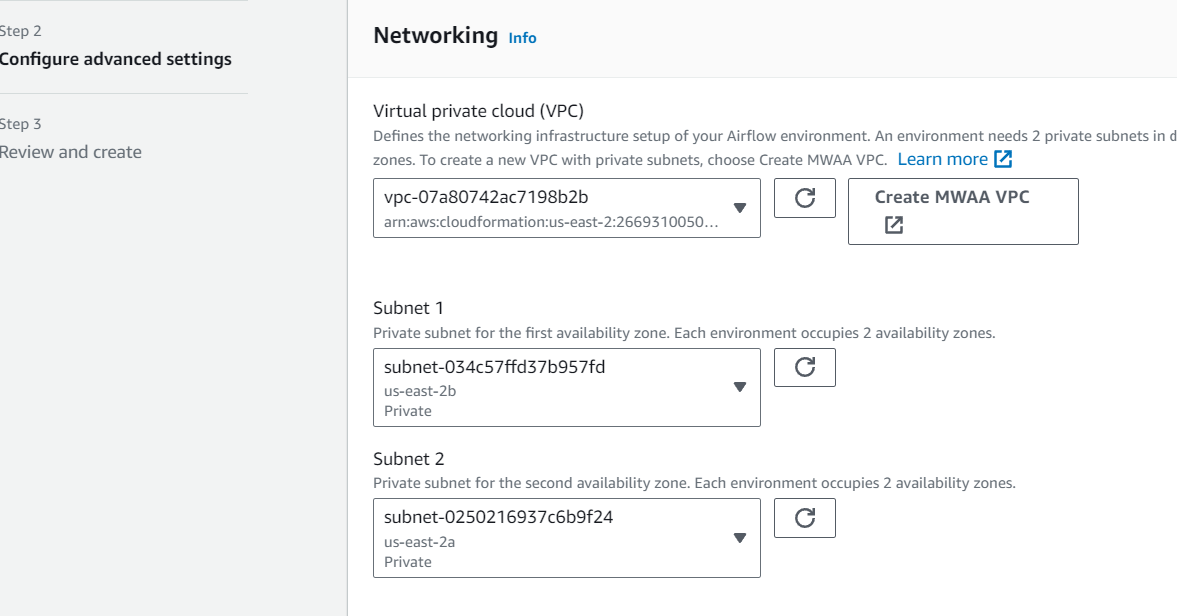
The configuration steps in creation of MWAA are:

1. Add the DAG and requirements.txt to S3 and add the path

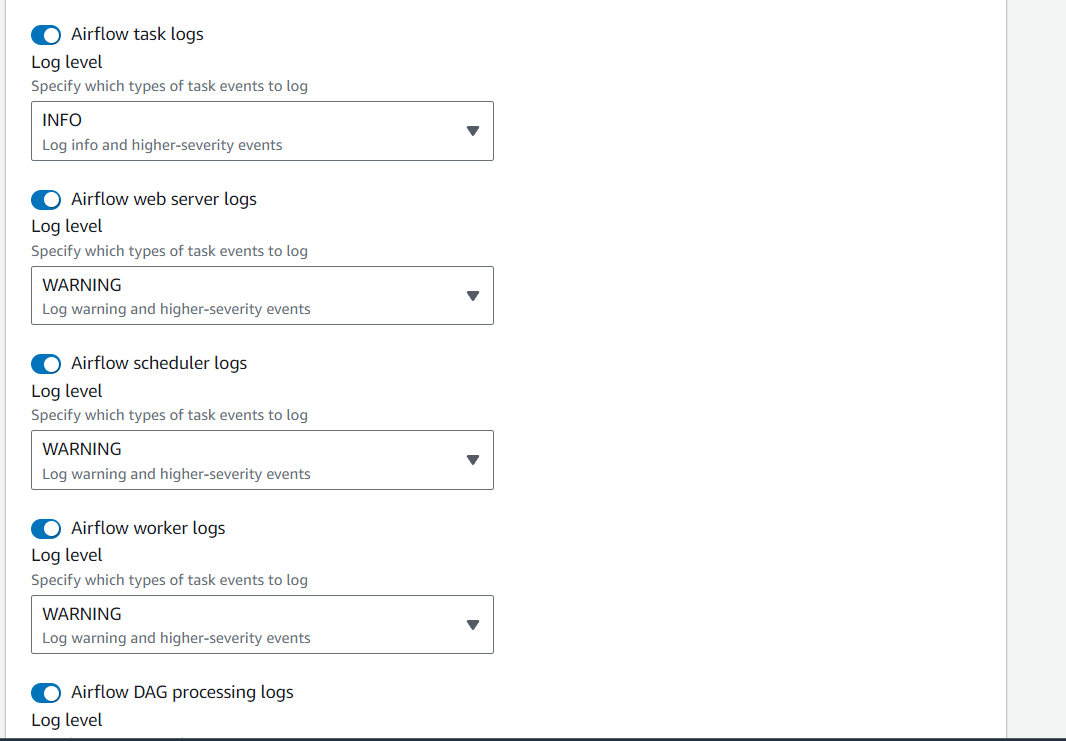


1. Create a new VPC





1. Create new security group
2. Environment class: choosing small
3. Monitoring : add all 4



1. Create a new role for this and add respective access (S3)
2. Other configurations check for need and define and **Create Environment** (can take 15-30 min)

# Data Visualization and Storytelling: